AD-A239 465

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
S ELECTE
AUG 19 1991
D
D

THESIS

A METHODOLOGY FOR PRODUCING AND TESTING A
GENESIL SILICON COMPILER DESIGNED
VLSI CHIP WHICH INCORPORATES
DESIGN FOR TESTABILITY

by

Brian L. Pooler

September 1990

Thesis Advisor: Herschel H. Loomis, Jr.

Approved for public release; distribution is unlimited

91 8 16 024

91-08090

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
UNCLASSIFIED	

2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT
	Approved for public release;
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	distribution is unlimited

4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION	6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION
Naval Postgraduate School	AS	Naval Postgraduate School

6c ADDRESS (City, State, and ZIP Code)	7b ADDRESS (City, State, and ZIP Code)
Monterey, California 93943-5000	Monterey, California 93943-5000

8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) METHODOLOGY FOR PRODUCING AND TESTING A GENESIL SILICON COMPILER DESIGNED VLSI CHIP WHICH INCORPORATES DESIGN FOR TESTABILITY

12 PERSONAL AUTHOR(S)
POOLER, Brian L.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1990 September	170

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.

17	COSATI CODES		18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Design for testability; VLSI; Genesil Silicon
			Compiler; Automatic Test Generation; DAS 9100;
			DV550

19 ABSTRACT (Continue on reverse if necessary and identify by block number) Testability issues concerning the need for including Design for Testability (DFT) techniques in VLSI designs are discussed. Types of fault models, the use of fault simulation and the DFT techniques of Scan Path and Built-in Test are described. An engineering methodology that uses the Genesil Silicon Compiler to produce a VLSI design, DFT_CHIP, which utilizes the DFT Scan Path techniqe is presented. Included are the procedures for using Genesil's simulation, timing analysis and automatic test generation features. The steps taken to fabricate the DFT_CHIP design through MOSIS are discussed. The methodology used to test the fabricated DFT_CHIP design on the Tektronix DAS 9100 tester is described. Appendix A and Appendix B provide copies of the Genesil check functions written for use during simulation on the DFT_CHIP design. Appendix C specifies the Genesil timing information for the DFT_CHIP design. Appendix D lists the conversion program which translates Genesil produced test vector files to the file format used during testing on the Tektronix tester.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
LOOMIS, Herschel H. Jr.	408-646-3214	EC/Lm

DD Form 1473, JUN 86   Previous editions are obsolete   SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

i

A Methodology for Producing and Testing a
Genesil Silicon Compiler Designed VLSI Chip Which
Incorporates Design for Testability

by

Brian Lee Pooler
Captain, United States Marine Corps
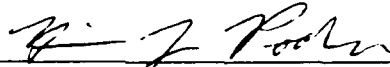B.S.E.E., United States Naval Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

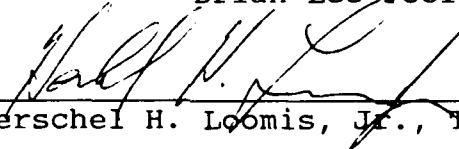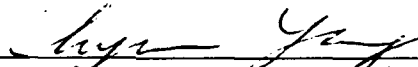NAVAL POSTGRADUATE SCHOOL
September 1990
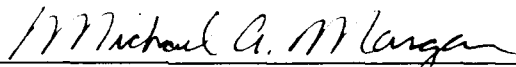
Author: _____
Brian Lee Pooler

Approved by: _____
Herschel H. Loomis, Jr., Thesis Advisor

_____
Chyan Yang, Second Reader

_____
Michael A. Morgan, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

Testability issues concerning the need for including Design for Testability (DFT) techniques in VLSI designs are discussed. Types of fault models, the use of fault simulation and the DFT techniques of Scan Path and Built-in Test are described. An engineering methodology that uses the Genesil Silicon Compiler to produce a VLSI design, DFT_CHIP, which utilizes the DFT Scan Path technique is presented. Included are the procedures for using Genesil's simulation, timing analysis and automatic test generation features. The steps taken to fabricate the DFT_CHIP design through MOSIS are discussed. The methodology used to test the fabricated DFT_CHIP design on the Tektronix DAS 9100 tester is described. Appendix A and Appendix B provide copies of the Genesil check functions written for use during simulation on the DFT_CHIP design. Appendix C specifies the Genesil timing information for the DFT_CHIP design. Appendix D lists the conversion program which translates Genesil produced test vector files to the file format used during testing on the Tektronix tester.

iii

Accesion For

NTIS CRA&I
DTIC TAB
U announced
Justification

By
Distribution /

Dist

A-1

TABLE OF CONTENTS

v

I. INTRODUCTION

A. BACKGROUND

1. Testability Issues

Testability of VLSI (Very Large Scale Integrated) circuits deals with the issue of accomplishing measurements on a circuit to insure that it performs in the manner in which it was intended to perform. For a VLSI circuit to produce the "proper" or expected outputs several factors must be looked at. First, the circuit logic must be designed correctly to produce the desired outputs for a given set of inputs. Secondly, the chip must be physically fabricated properly so as to correctly implement that logic for which it was designed. Finally, the circuit should retain correct functionality over time by having stable operating characteristics. If proper engineering techniques were used to formulate the logic design for the circuits of a VLSI chip, the major testability issue centers on the ability to completely evaluate the physical functioning of circuits/gates internal to the chip. **Design for Testability** (DFT) is an approach to designing VLSI chips/circuits to better ensure that individual internal components can be accessed for testing purposes.

With future improvements of VLSI technology, the complexity (in terms of the number of components, gates, or circuits) of VLSI chips will continue to increase. As this

1

complexity increases, the degree of difficulty experienced during the testing of VLSI circuits will increase correspondingly. "Conventional" testing relies on adding additional mechanical means for testing such as extra input/output (I/O) pins for more test points, improving test fixtures, using additional testing probe points with a "bed of nails" etc., and suffers by being able to conduct testing only when the part is removed from the system it operates in [Ref. 1:p. 48]. In contrast, DFT relies on the addition of logic circuits internal to the chip to help facilitate testing and circuit accessibility, and DFT techniques can be used to design systems which allow in-system testing of parts [Ref. 1:p. 81].

Conventional testing methods have become inadequate primarily due to their inability to access internal circuit components and their need to feed signals through a test interface involving many I/O pins [Ref. 1:p. 57]. As VLSI chips become more dense they require extra pins for normal I/O operations thus leaving fewer pins which can be used for testing purposes. Also, with the increased miniaturization of VLSI chip circuitry, the number of pads available to connect to I/O pins has not kept pace with increases in the number of transistors within a chip [Ref. 1:p. 59]. When chip periphery lengths grow by Δl the area available for transistors grows by $(\Delta l)^2$ but the space available for pads grows by only $4\Delta l$. If this transistor growth requires more I/O pins and pad growth can not keep up, the pins available for testing will decrease.

2

The need for considering the inclusion of DFT techniques during chip design can be predicated largely on one factor: cost [Ref. 2:p. 100]. The need to insure reliability in a chip is self-evident and only testing can help insure reliability. If the cost of testing a complex VLSI chip is too great, an otherwise desirable logic design might not be produced. Although VLSI per circuit fabrication costs have been decreasing, the per circuit testing costs have increased as a percentage of the total chip cost as chips have become more complex [Ref. 1:p. 15]. Additionally, the costs associated with a user finding/using defective chips mandates that adequate testing be done prior to chips being distributed for use. To both lower the costs of performing testing and to allow a degree of testing in otherwise conventional testing prohibitive circuits, the inclusion of DFT criteria has emerged as a growing requirement in VLSI design.

2. **Fault Models**

The goal of DFT is to find ways to make testing easier, less costly, and more efficient to implement. By adding additional circuitry to the chip, DFT adds to the observability and controllability of the system. **Controllability** can be defined as the degree to which a node internal to a circuit can be set to a given logic level [Ref 3:p. 97]. In contrast, **observability** refers to the ability to observe the logic level of a given internal node via an output from the design [Ref. 3:p. 97]. The degree to which a chip can be

3

tested is highly dependant upon its degree of controllability and observability. By increasing the controllability and observability of a circuit, the testability, in terms of finding out if the circuit is "fault free", is increased.

A circuit is said to contain **faults** if it exhibits failures which cause deviations from the specified performance behavior [Ref 4:p. 1]. Two major classes of faults are design faults which manifest themselves as improper connections within the VLSI circuit and physical circuit defect faults such as those caused by manufacturing problems or wear out during chip operation [Ref. 4:p. 1]. Physical defects include open contacts, broken lines, faulty transistors, and shorts between parts of the circuit [Ref. 4:p. 1]. Photolithography errors during the manufacturing of VLSI circuits, such as alignment problems, mask failures, or unintended or missing connections, are major contributors to these physical defects [Ref. 5:p. 693]. Wear out failures can occur due to such things as metal starting to corrode or metal migration due to the presence of high current densities [Ref. 5:p. 695]. **Fault models** are used as a means for describing what the effects are of a particular type of circuit physical failure.

Fault models can include modeling faults down to physical defects at the individual transistor/switch level, but often fault models only consider faults down to the logic gate level. The advantage to using a logic gate level fault model is that this type of model can represent faults for many

4

different technologies. The **stuck-at fault** model is this type of logic gate level model and as such is the lowest level of modeling that is technology independent.[Ref. 6:p. 40]

The stuck-at fault model is based on assuming that physical defects causing faults will result in input or output lines of logic gates being permanently stuck-at logic level 0 or 1. As an example of how a stuck-at fault could be manifested in a CMOS circuit consider the inverter of Figure 1.1. If the line at point A is inadvertently shorted to ground then the output of the inverter will be stuck-at 1 (S-A-1) no matter what the input to the inverter is. If the line at point B is inadvertently broken then the circuit will produce the correct logic level 1 output when a logic level 0 is the input and the p-type transistor conducts. However, if a logic level 1 is the next input after a logic level 0 input the n-type transistor is not turned on because of the broken line. Instead, the output will remain at logic level 1 for a period of time which depends on leakage currents. If the inverter is included in a circuit which receives a high speed stream of inputs consisting of both logic level 1's and 0's then the time for the leakage current dissipation may be longer than the time between inputs of logic level 1's. If so, the output will appear as a permanent S-A-1 fault.[Ref 7:pp. 7, 8]

To illustrate how to test for stuck-at faults, the AND gate of Figure 1.2 is considered. If this gate has a S-A-1 fault at the gate's A input as shown, the gate will always

5
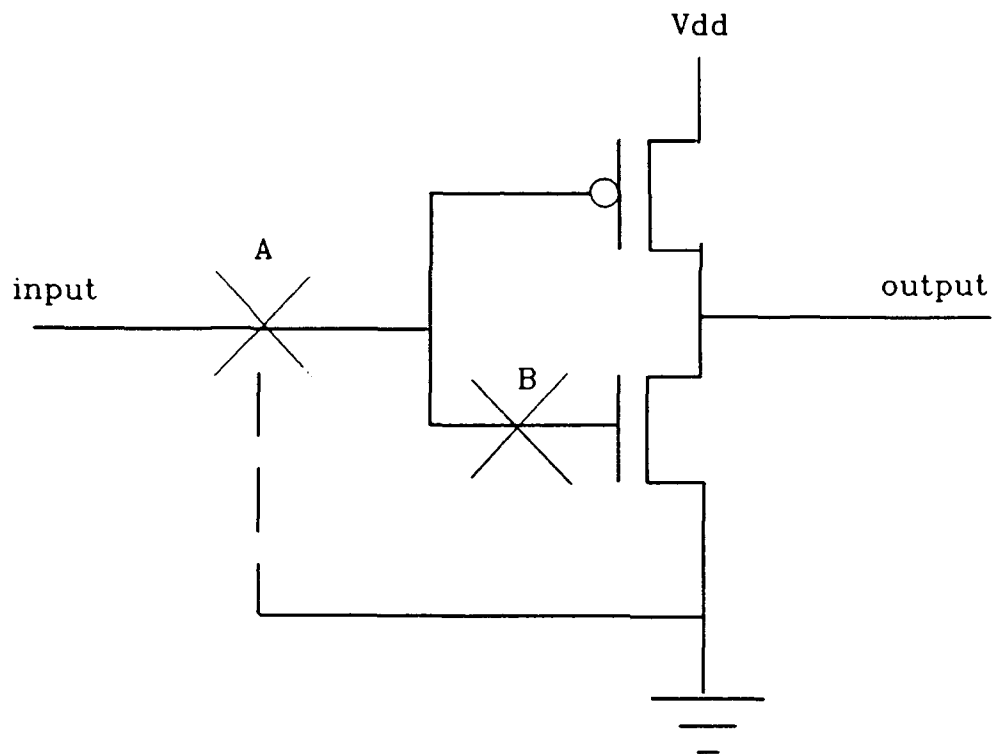
Figure 1.1   CMOS Inverter Stuck-at Faults After Ref. 7.

S−A−1

Using inputs of A=0 and B=1 will

check for a S−A−1 fault at input A
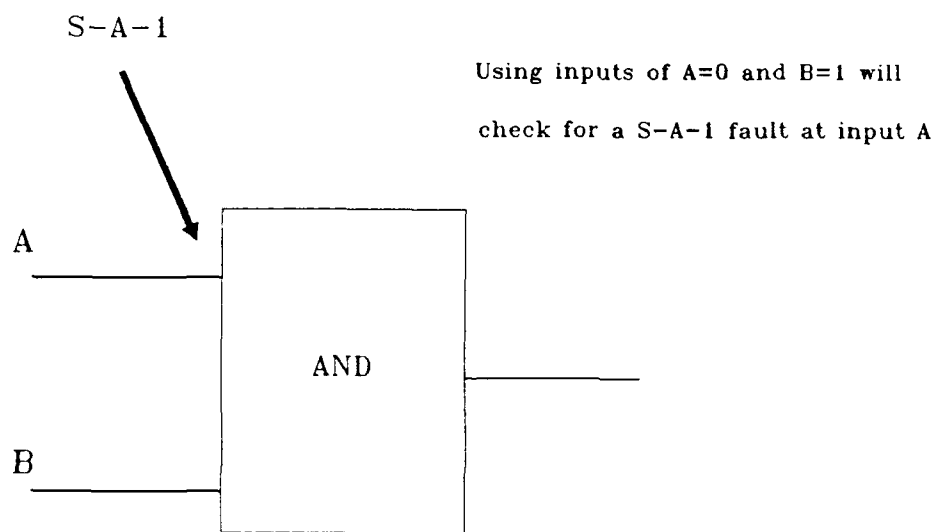
Figure 1.2   AND Gate Stuck-at Fault Testing

perceive line A to be at logic level 1, even if a logic level 0 is actually applied. To test for this S-A-1 fault for line A, note that the A,B input line pairs (0,0), (1,0), and (1,1) will all produce correct outputs on line C. Therefore, these inputs can not produce a result which indicates the presence of the S-A-1 fault. However, if input (0,1) is used the output should be logic level 0 but will instead be logic level 1 due to the S-A-1 fault at input A. Thus, the input pattern (0,1) is a test for a line A S-A-1 fault.

Each logic gate having a total combination of n input/output lines has a possibility of 2n different single (one at a time) stuck-at faults (i.e., each input or output line can exhibit either a S-A-1 or a S-A-0 fault). One problem in testing is to design test vector inputs which can detect these 2n stuck-at faults. The AND gate of Figure 1.3 shows that certain input patterns can determine the presence of more than one stuck-at fault. As an example, an A to D input line pattern of (1,0,1,1) which produces an output of logic level 1 means that either input line B is S-A-1 or output line E is S-A-1. As shown in Figure 1.3, only five test vectors are needed to completely test the proper functioning of this gate. Note that a test vector which produces an erroneous result will not be able to specify which stuck-at fault exists since each test vector covers more than one stuck-at fault case. Instead, the results from the application of the test vector will only indicate that the gate has

7
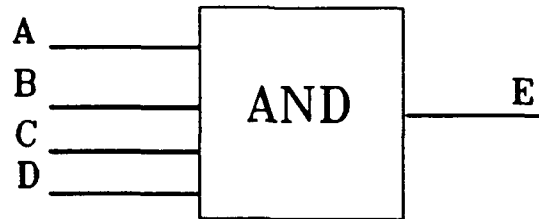
Input Faults		Pattern to Detect Fault			
		A	B	C	D
A	S-A-0	1	1	1	1
A	S-A-1	0	1	1	1
B	S-A-0	1	1	1	1
B	S-A-1	1	0	1	1
C	S-A-0	1	1	1	1
C	S-A-1	1	1	0	1
D	S-A-0	1	1	1	1
D	S-A-1	1	1	1	0

Output Faults		Pattern to Detect Fault
		A B C D
E	S-A-0	1 1 1 1
E	S-A-1	any input combination that contains one or more 0 inputs

Patterns Needed to
Completely Test Gate

A	B	C	D
1	1	1	1
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure 1.3   Test Vectors Needed for Stuck-at Fault Detection

a fault in it. However, this information alone is often all that is sought since then the VLSI circuit is known to be bad.

Although basically technology independent and widely used, the stuck-at fault model does have several problems. Among these are:

1. Only a _single_ stuck-at fault is assumed to be present at a time in the VLSI circuit for this model type. Since more than one stuck-at fault could actually be present in a circuit, the stuck-at fault model does not accurately represent the true range of conditions possible [Ref. 8:p. 97].

2. The stuck-at fault model does not take into account high speed "AC" or "dynamic" type faults [Ref. 1:p. 403].

3. Certain types of faults such as bridging faults, involving shorts between lines, and floating-gate type faults can not be completely handled by the stuck-at fault model [Ref. 1:p. 404].

To overcome these problems other models such as the bridging fault model (a gate level model), and the stuck-open and stuck-on fault models (both transistor level models) can be used [Ref. 4:pp. 10, 11]. Today however, the single stuck-at fault model is still the model which is used most widely for testability and fault simulation purposes [Ref 3:p. 95]. The work done for this thesis is based on using a single stuck-at fault model.

3. Fault Simulation

In order to test a VLSI circuit a set of test vectors must be applied to the circuit and then the output results must be compared to the desired results (normally obtained from logic simulation). The purpose of **fault simulation** is to determine, for a specific set of test vectors, which faults in

9

the circuit are detected. Fault simulation normally involves a fault simulator introducing single stuck-at faults, one at a time, into gate level models of the circuit [Ref. 6:p. 37]. By then applying the test vectors to the simulated circuit, the fault simulator determines if the fault was or was not tested/detected. The result of fault simulation is a determination of the percentage of faults that were tested/detected with a given set of test vectors out of the total number of faults introduced by the fault simulator. Just because a fault is not detected does not mean that it is untestable; rather it just means that the given set of test vectors used could not detect it [Ref. 8:p. 97]. **Fault coverage** is the total number of faults, expressed as a percentage, that can be detected for a given set of test vectors as compared to the total number of all possible faults [Ref. 1:p. 335].

It is important to make a distinction between the two reasons and types of testing that is performed on a chip. Functional testing is used to validate the operational characteristics of a chip. Functional test vectors are applied to the chip and the outputs are observed to determine if they provide the results desired for a given input or sequence of inputs. In contrast, a set of test vectors which achieve maximum fault coverage is applied to a chip to determine if faults arising from the fabrication process are present. Therefore, the maximum fault coverage test vector set is not intended to check for proper design functionality

of the chip, but rather to check for manufacturing errors which result in faults that would preclude the chip from performing in its intended manner. Although technically possible, a set of maximum coverage test vectors is not normally used to check chip functionality because of the extreme complexity of determining the "proper" outputs (in terms of functionality) for the sequence of inputs that provides maximum fault coverage.

The question most frequently asked when performing testing is often "How much testing is enough?" A correlation has been found between fault coverage and average quality level (AQL), where AQL is a measurement of the percentage of defective parts found by users. Fault coverage must be very high to obtain an acceptably low AQL. A fault coverage of 50 to 70 percent produces approximately a five percent AQL, 90 percent fault coverage a three percent AQL, and it takes up to 99 percent fault coverage to get a 0.1 percent AQL. For an application specific integrated circuit (ASIC) 99.9 to 99.99 percent fault coverage is considered mandatory by many experts today.[Ref. 6:p. 37]

Of concern to a manufacturer of VLSI circuits is the defect level which is present for a given yield and percentage of fault coverage on a chip. A derivation, using the following steps as found in Ref. 8, produces an equation for the defect level in terms of these parameters:

1. Use the stuck-at fault model and assume all stuck-at faults are independent from each other.

2. Let P_n be the probability of a single stuck-at fault occurring on a chip which has n such possible faults. Let yield (Y) equal the probability of a good chip. Then since each fault is independent

$$Y = (1-P_n)^n. \tag{1.1}$$

3. Let A represent the case where the chip has no stuck-at faults on it. Let B represent the case where it is determined that no stuck-at faults are present in any of m sites that have been tested. Then the probability of B found from testing m out of n total possible faults is

$$P(B) = (1-P_n)^m. \tag{1.2}$$

4. The probability of A (probability of a good chip) given that B occurred (m of the sites were tested without finding a fault) is

$$P(A|B) = \frac{P(A \cap B)}{P(B)}. \tag{1.3}$$

5. Now P(A∩B) is P(A) = no faults on the chip and P(B) = no faults at m sites on the chip. If there are no faults on the chip there are also no faults at any set of m sites on the chip. Therefore,

$$P(A \cap B) = P(A) = (1-P_n)^n. \tag{1.4}$$

6. Let DL = defect level = probability that a defective chip is manufactured and sent to a user. DL = 1 minus the probability that a good chip is sent. Therefore,

$$DL = 1-P(A|B) = 1-\frac{P(A \cap B)}{P(B)} = 1-(1-P_n)^{n-m}. \tag{1.5}$$

7. Substituting equation (1.1) into equation (1.5) gives

$$DL = 1-Y^{(\frac{n-m}{n})} = 1-Y^{(1-\frac{m}{n})}. \tag{1.6}$$

8. Let T = fault coverage. Note that T = m/n (i.e., testing m of n possible stuck-at faults). Then the defect level can be expressed as

$$DL = 1-Y^{(1-T)}. \tag{1.7}$$

The graphical consequences of equation (1.7) can be seen in Figure 1.4. Note that even for high yields a very high fault coverage percentage is needed to get an acceptably low defect level. In a study conducted by Motorola, Published in IEEE Design and Test (April, 1985), using an actual manufacturing line with 50 percent yield, a fault coverage of 97 percent still produced a one percent defect level [Ref. 6:p. 40]. Figure 1.4 shows how equation (1.7) produces results which closely follow those obtained from this study even with the assumptions made about only independent stuck-at type faults being present on the chip.

The obstacle to using the fault simulation process is the time it takes to run a simulation evolution. Fault simulation time tends to rise exponentially as circuits become more complex [Ref. 9:p. 59]. This can lead to the inability to perform fault simulation, due to time constraints alone, on complex chips. Parallel or concurrent simulation algorithms which allow more that a single stuck-at fault to be examined during a given time period are alternative methods which can accelerate the simulation process [Ref. 6:p. 41]. Another method is via statistical fault grading which attempts to extrapolate the information gathered from a limited number of test vectors to predict overall fault coverage by factoring in information of controllability and observability. Although obviously not as accurate as complete fault simulation, statistical fault grading appears to be able to come within a
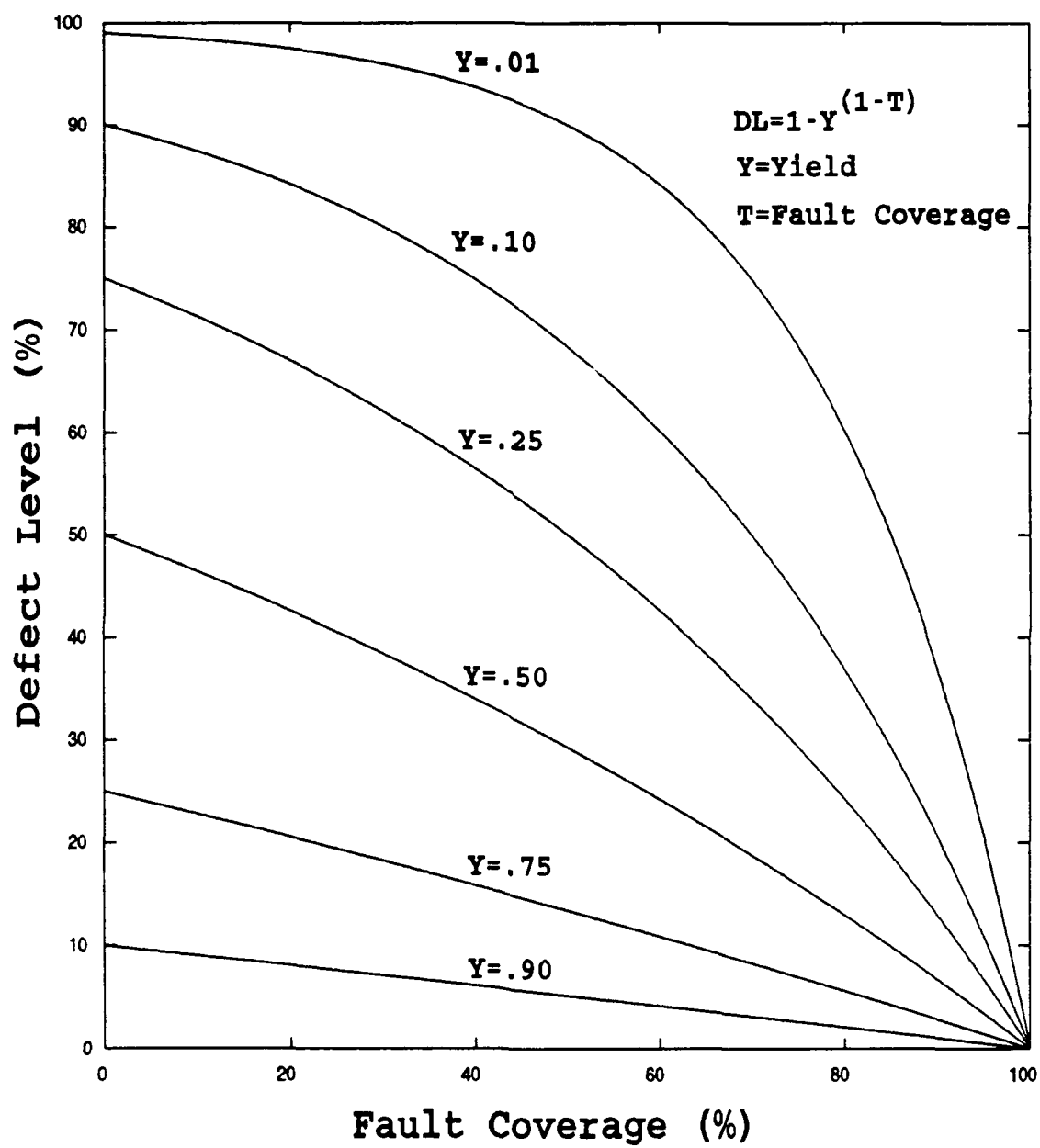
Figure 1.4   Defect Level versus Fault Coverage

few percent of the accuracy obtainable using complete fault simulation [Ref. 9:p. 59].

B. **THESIS OVERVIEW**

The primary goals of this thesis are twofold: first, to investigate the incorporation of DFT on a VLSI chip implemented on the Genesil Silicon Compiler; second, to use Genesil to design a chip with DFT features, have it fabricated, and physically test the chip. Through examination of the steps taken in working with an actual chip, this thesis will concentrate on the methodology for both incorporating DFT techniques into Genesil designs and for testing a fabricated Genesil implemented chip on a commercially available tester. Chapter II will describe the DFT techniques of Scan Path and Built-in Test. It will provide a comparison of the relative advantages and disadvantages of both these techniques. Finally, it will discuss the manner in which Genesil implements the Scan Path technique chosen for use on the chip which was fabricated was fabricated. Chapter III is concerned with the methodology process used during the design, simulation testing, test vector fault grading and automatic test vector generation for a 16-bit correlator chip produced using the Genesil Silicon Compiler. It will also include a complete functional description of this chip. Chapter IV provides information pertaining to the process of getting the chip fabricated via MOSIS and examines the methodology used and results obtained during testing conducted on the fabricated

chip.   Chapter V provides a summary of and draws conclusions from the research done during the course of this thesis.   The appendices provide copies of the programs and functions written to perform simulation on the correlator chip, information obtained about timing characteristics of the chip, and a copy of the program written to convert test vectors from the format provided by Genesil to that needed by the commercial chip tester.

II. DESIGN FOR TESTABILITY TECHNIQUES

A. BACKGROUND

The need for a high degree of fault coverage to insure a quality product has made it necessary to consider DFT issues while developing new VLSI designs. If internal circuit nodes cannot be initialized to needed test vector values (an issue of controllability) and/or the results of the tests cannot be seen (an observability issue) then the fault coverage possible will not be as high as is considered desirable. Only by including DFT circuit logic in a chip can observability and controllability be increased enough to raise fault coverage to acceptable levels for complex VLSI chips.

Two major techniques have evolved for incorporating DFT into chip design. The first is the **scan path** technique which involves the serial introduction of externally generated test vectors into specific internal nodes of a chip to enhance controllability and the serial extraction of internal node values from the chip to enhance observability. **Built-in Test** or **Self-test** is the second technique. This technique uses additional circuitry on a chip to produce test vector patterns internally and may include circuitry to simplify the analysis of the test results. These two techniques can be used independently or both can be included in a single chip design.

Complementary Metal Oxide Silicon (CMOS) chips designed using Genesil may include either or both of these techniques. Genesil provides a **Testability Latch Block** which may be included in a chip design for the major purpose of increasing controllability and observability. For parallel datapath designs three configurations of the Testability Latch Block are available:

1. The basic configuration which uses a single shift register to serially enter or retrieve data.

2. The generator configuration which has the attributes of the basic configuration as well as including circuitry for pseudorandom test sequence generation.

3. The signature configuration which has the attributes of the generator configuration plus signature analysis logic circuitry.

The first configuration is used during implementation of the scan path technique while the latter two configurations make use of Linear Feedback Shift Registers to help implement the Built-in Test technique.[Ref. 10:p. 24-2]

B. COMPARISON OF THE SCAN PATH AND BUILT-IN TEST TECHNIQUES

1. Scan Path

The scan path technique is a conceptually easy approach to including DFT into a chip design. Scan path designs create a situation whereby access may be gained to the internal circuitry of a VLSI chip using a minimal number of chip pins devoted to testing. This technique enhances the observability and controllability of internal nodes which would otherwise be inaccessible from the periphery of the

18

chip. The scan path technique accomplishes this by partition-
ing a design into smaller subsystems which can then be tested
to a higher degree and in an easier manner than the original
system as a whole. A scan path is nothing more than a serial
channel through which data can be shifted to reach flip-flops
which provide a desired initialization state to specific
internal nodes of the chip. Through the shifting process,
specific values can be latched into the internal nodes prior
to the start of a test ("scanned input") and the results can
be shifted out after the completion of a test ("scanned
output") [Ref. 1:p. 103]. Figure 2.1 illustrates the manner
in which a generic circuit might be broken into smaller
subsystems via use of the scan path technique. Thus, scan
path solves the controllability problem via its ability to
shift in data to internal nodes and solves the observability
problem through its ability to access the results of tests by
shifting them out of the chip.

The scan path technique is often used in connection
with the flip-flops that already exist in a VLSI design due to
the presence of sequential circuitry. This involves connect-
ing the flip-flops which provide "memory" to the sequential
circuit together in a controllable serial shift register type
fashion. Then by using "switches" controlled by test signals,
the flip-flops can be changed from the "normal" sequential
circuit mode to a "shift register mode" in which the flip-
flops form a shift register through which specific test

19
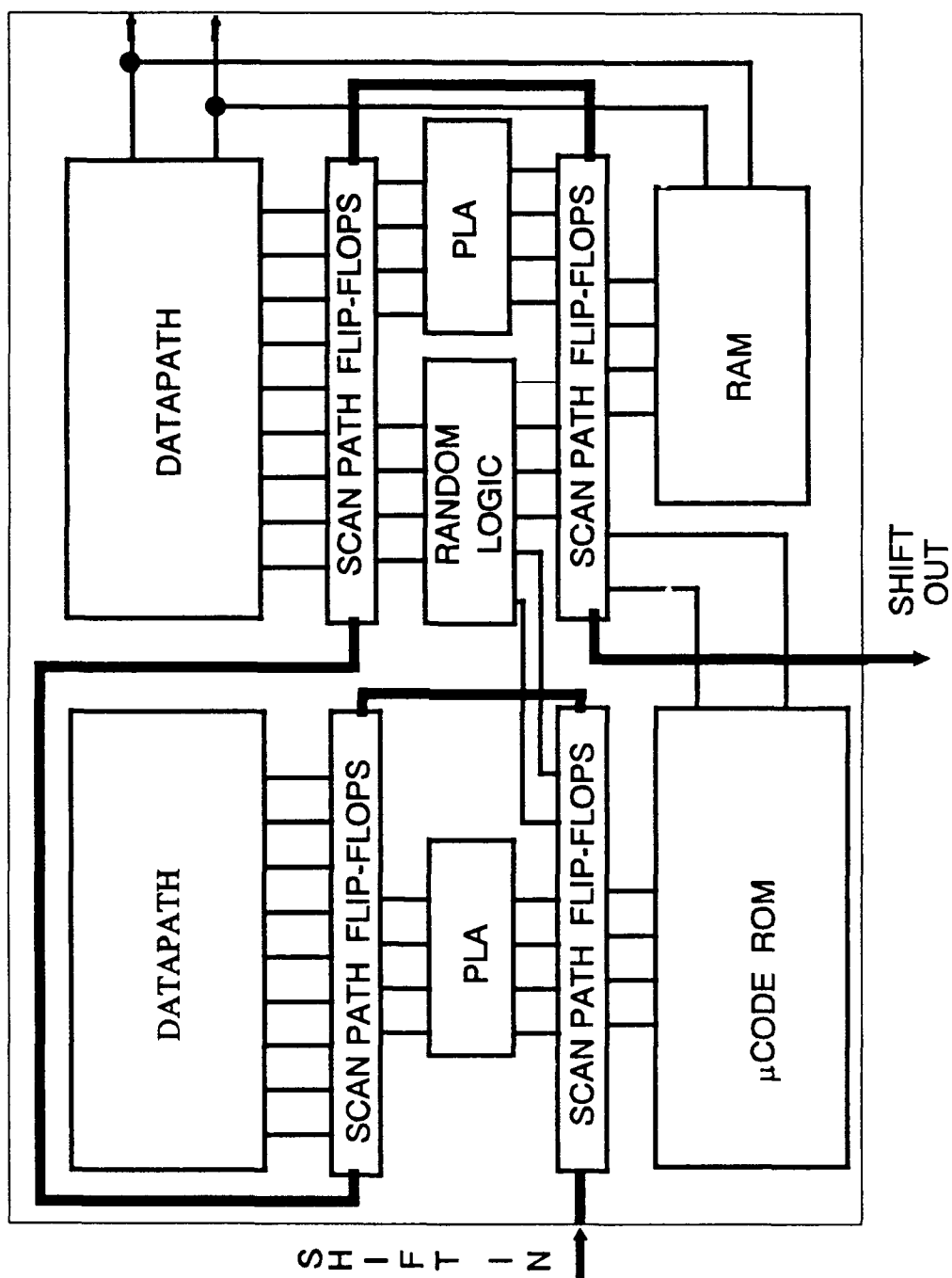
Figure 2.1 Circuit Broken into Smaller Subsystems
After Ref. 7

20

vectors can be introduced into particular locations of the circuit [Ref. 1:p. 108]. This makes possible the testing of the circuit. Figure 2.2 illustrates the process of introducing a scan path into a sequential circuit.

There are several individual and slightly different approaches which make use of the scan path idea. The actual **Scan Path** method chains together master-slave D type flip-flops to make the needed shift registers. These D type flip-flip-flops consist of two latches controlled by a single clock signal. The clock signal for the first latch goes through an inverter to become the clock signal for the second latch. The disadvantage to this approach is that if the input to the D flip-flop changes at nearly the same time that the clock does or if the output of the second latch feeds back through combinational logic to become the input of the first latch then a race condition could exist.[Ref. 2:p. 105]

To overcome this potential race problem an approach called **Level Sensitive Scan Design** (LSSD) was developed. It utilizes the same basic idea as Scan Path for moving test vectors into and out of the circuit but uses two separate clocks, each controlling a separate latch from a two latch pair. By using two separate clocks to control a latch pair element of the shift register, the LSSD technique overcomes the potential race problem present for the Scan Path technique.[Ref. 2:p. 105]

INPUTS

OUTPUTS

COMBINATIONAL
LOGIC

SCAN IN

SWITCH CONTROL

"MEMORY"
FLIP-FLOPS

SCAN OUT
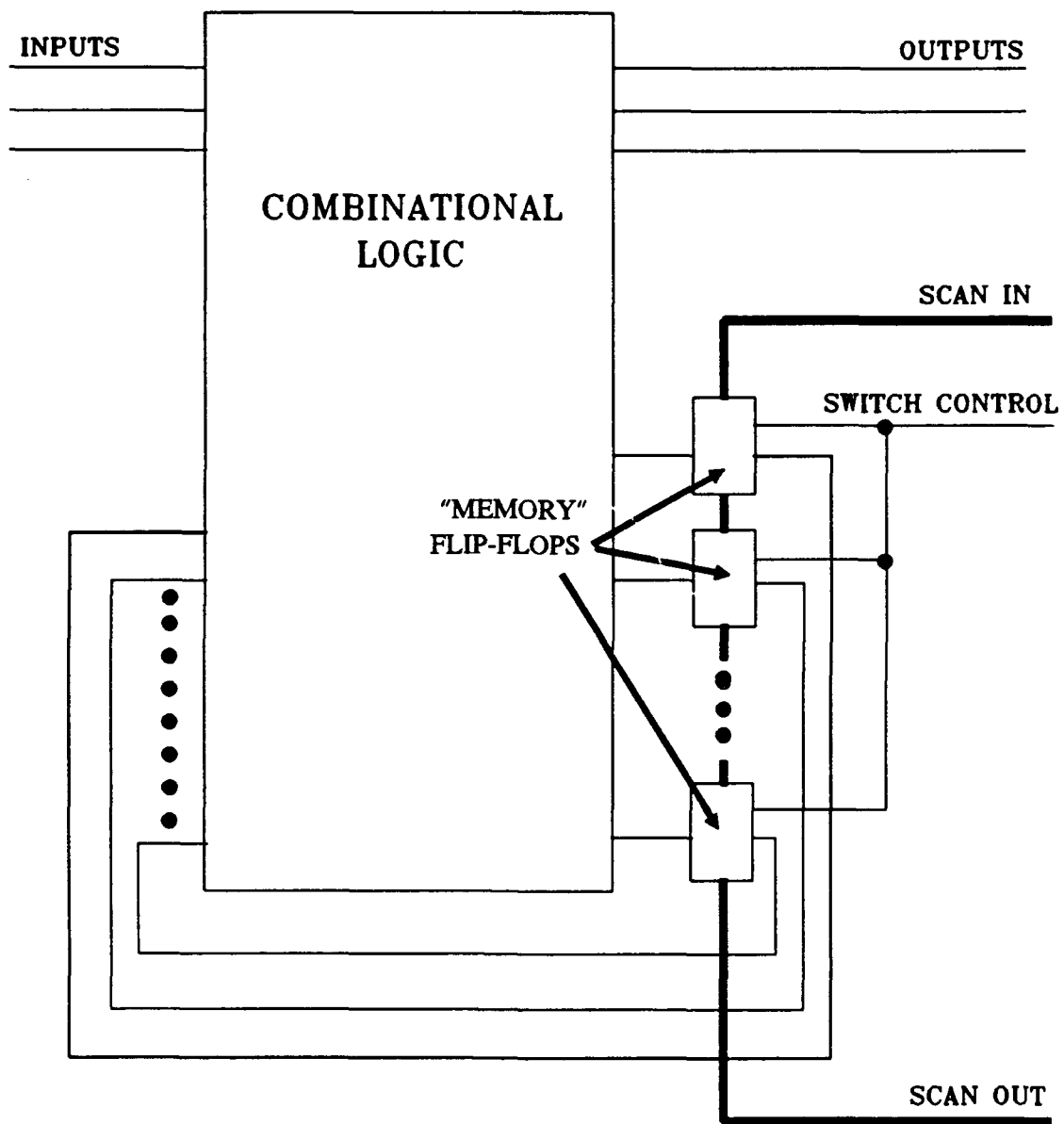
Figure 2.2   Using a Scan Path in a Sequential Circuit

Finally, a related method which has been developed is the **Scan Set** method. This technique varies from straight scan path methods in that although it uses shift registers, they are <u>not</u> located in the data path. Instead, the shift registers are placed adjacent to the circuit's own sequential circuit latches. There are two advantages to this method. First, a designer can determine exactly which of the sequential circuit latches he desires to have the ability to set if the ability to set all latch points is not needed or desired. Secondly, Scan Set sampling can occur <u>during</u> the sequential circuit clocking period thus providing a snap shot of the sequential circuit during operation.[Ref. 2:p. 106]

2. Built-in Test

The Built-in Test (BIT) technique is inherently more complex than the scan path technique. BIT involves a tradeoff of additional chip circuitry above that used in the scan path technique against the ability to internally generate test vector patterns and compact test result responses. This facilitates the testing procedure by moving a portion of the test process from off to on chip. The scan path technique needs to scan-in a test vector and or scan-out a test result for each test cycle. In contrast, BIT reduces the need to scan-in a new test vector for each test cycle since it internally generates its own test vector patterns after an initialization process. Additionally, BIT can reduce the need to scan-out test results each test cycle by encoding the

23

results in a more compact form which need be looked at only after the completion of a large number of test cycles. The combination of these two effects results in BIT allowing multiple test cycles to proceed at full system speed with only the overhead of single initialization and final result gathering phases to slow it down.[Ref. 1:pp. 169, 170]

The use of a structure called a **Linear Feedback Shift Register** (LFSR) is the main method which has been developed to provide internally generated test vector patterns. The patterns which are produced are normally deemed to be **pseudo-random** in nature since they follow no apparent order from pattern to pattern and meet some basic statistical tests for randomness. In reality they are not random, thus the designation pseudo, but rather follow a predetermined sequential order which depends both upon the implementation configuration of the LFSR and upon the initialization values place in the flip-flops of the LFSR.[Ref. 1:p. 172]

The basic structure of a LFSR as implemented by Genesil is shown in Figure 2.3. It consists of an n-stage shift register with each stage position serving as a latching mechanism for a single bit. The multiplexer control line R determines whether the input for the bit zero stage position comes from the feedback loop or from the serial input line TIN. Initialization is provided to the LFSR by serially loading a desired value into each shift register bit stage position. Once initialization is complete the control line R
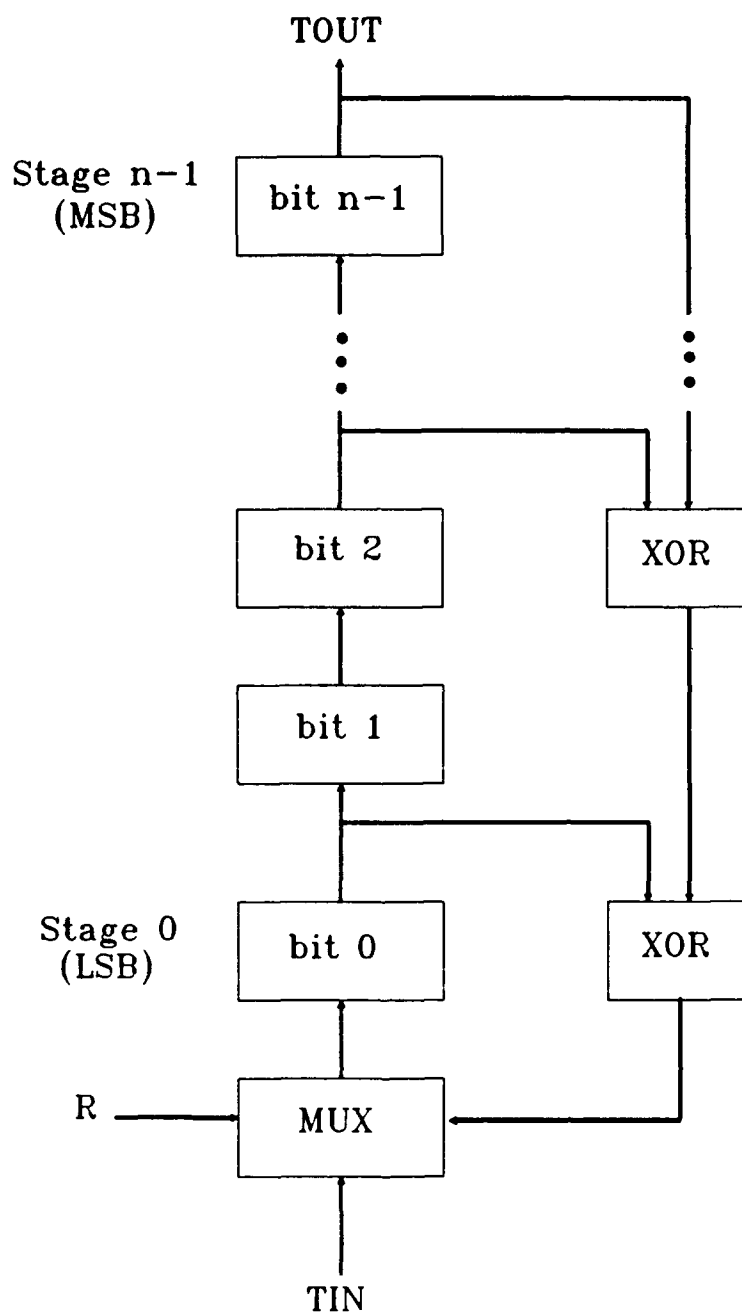
Figure 2.3   Linear Feedback Shift Register   After Ref. 10

can be set so that the bit zero stage position accepts values only from the feedback path. The changing contents of the LFSR bit stage positions will produce a pseudorandom sequence determined by the specific locations where exclusive-or (XOR) gates are found in the feedback path. By taking the output from each bit stage position and forwarding it to other circuit logic on the chip a sequence of internally generated test vector patterns is produced by the LFSR.

The locations where XOR gates are present in the feedback path is dependant on a constant called the LFSR polynomial which determines the number of terms between repetitions in the pseudorandom sequence [Ref. 7:p. 66]. It is desirable to have both the number of terms produced by the pseudorandom sequence generator be of maximum length and to utilize a minimum number of XOR gates in the feedback path. A maximum length sequence generator from an n-stage LFSR can produce 2^n-1 different sequence terms [Ref. 1:p. 175]. Reference 8 provides a detailed discussion on the development of LFSR polynomials which minimize the number of XOR gates needed to achieve a maximum length generator sequence for a given size n-stage LFSR.

A modified LFSR structure may be used to form a **signature analysis** register as shown in Figure 2.4. The signature analysis register is formed by adding an additional XOR gate as the last item in the feedback path to perform an XOR operation on the feedback path value and an incoming data bit

TOUT

Stage n−1
(MSB)   bit n−1

bit 2   XOR

bit 1

Stage 0   bit 0   XOR
(LSB)

                                              Incoming
                                              Data Bit
R   MUX   XOR

TIN
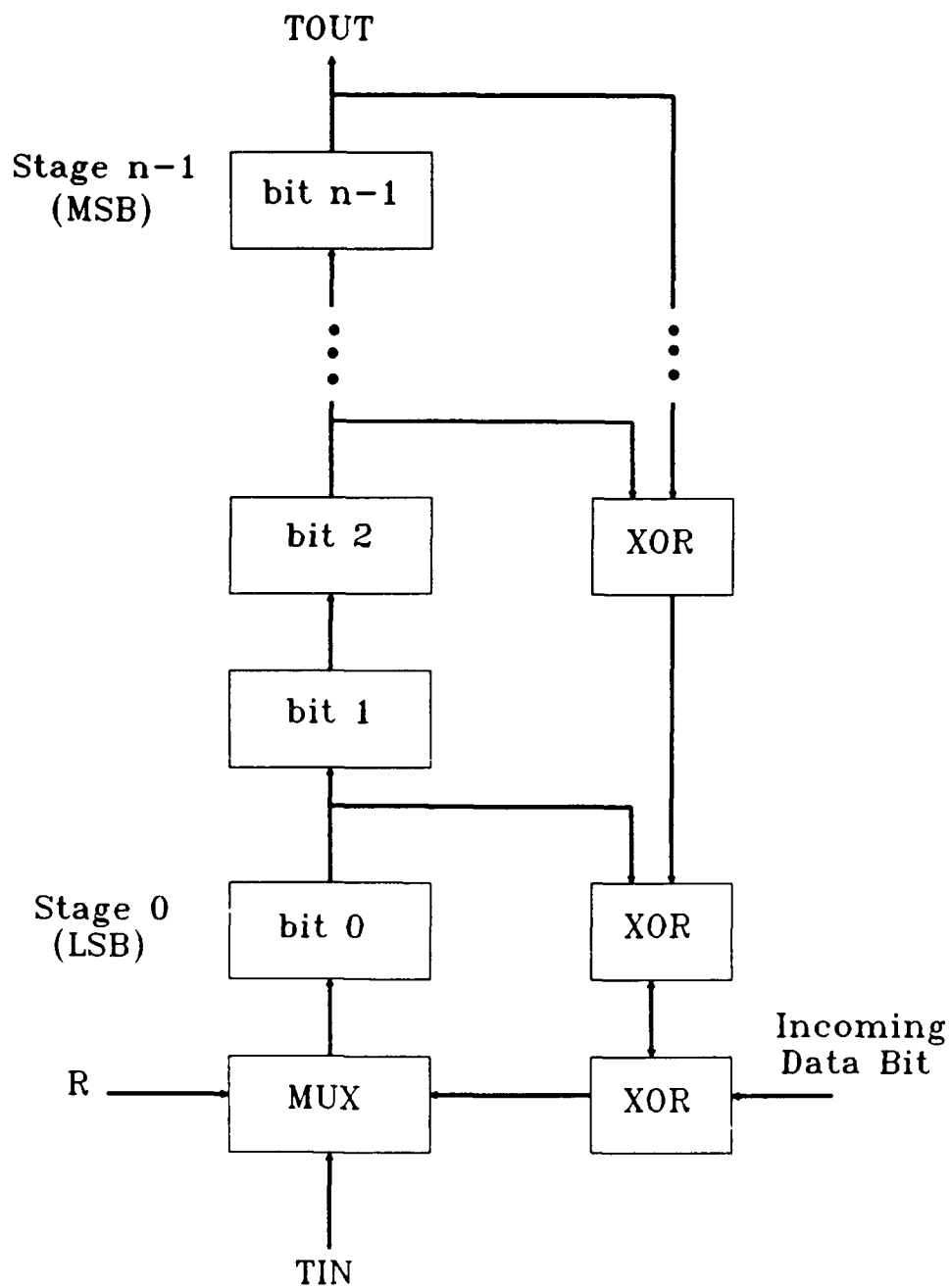
Figure 2.4   Signature Analysis Register

27

value [Ref. 8:p. 143]. Therefore, the pattern present in the signature analysis register is dependant on the initialization vector loaded serially into the shift register, the XOR gate structure of the feedback path, and the data sequence occurring for the incoming data bit. If the signal analysis register is initialized with a known value and the incoming data sequence is also known then the pattern which should be present in the shift register portion of the signal analysis register after a specific number of clock cycles can be determined through simulation.

Placing a signature analysis register on an output line of a circuit being tested allows the output test sequence results for a large number of consecutive tests to be compacted to a shorter cumulative sequence. Thus, the signature analysis register serves to encode the test result sequence. The encoded, cumulative results can be shifted out to compare against simulated results obtained by using the output data sequence from a properly functioning circuit. If the results vary between the two cases then a fault in the circuit has been detected.[Ref. 1:p. 177]

Built-in Logic Block Observation (BILBO) is a BIT method which is similar to the signature analysis approach. The BILBO structure differs from a signature analysis register mainly in that it has available additional XOR gates placed before each bit position to allow a total of n incoming test data bits to influence the encoded BILBO register value

[Ref. 8:p. 144]. Therefore, BILBO is well suited for compacting test results obtained from multiple internal nodes in a chip.

A general strategy for utilizing BILBO is shown in Figure 2.5 and Figure 2.6. Figure 2.5 shows the configuration used to test the first combinational logic block. Figure 2.6 shows the shifted configuration used to test the second combinational logic block. BILBO registers are ideal for this setup since if their test data bit input lines are held constant they can function as a LFSR test pattern generator and otherwise they can function as a test result compactor. By shifting the function performed by the two BILBO registers both sets of combinational logic can be adequately tested. [Ref. 2:p. 107]

3. Comparative Advantages and Disadvantages

a. Scan Path Advantages

The primary advantage in using the scan path technique is that it increases both the controllability and observability of otherwise inaccessible internal nodes. In doing so it raises the obtainable fault coverage level possible for a chip. Using a scan path makes possible the introduction of a minimal number of specifically designed test vectors to maximize the fault coverage [Ref. 11:p. 379]. Using these customized test vectors may in some circumstances reduce the total number of test vectors, as compared to the
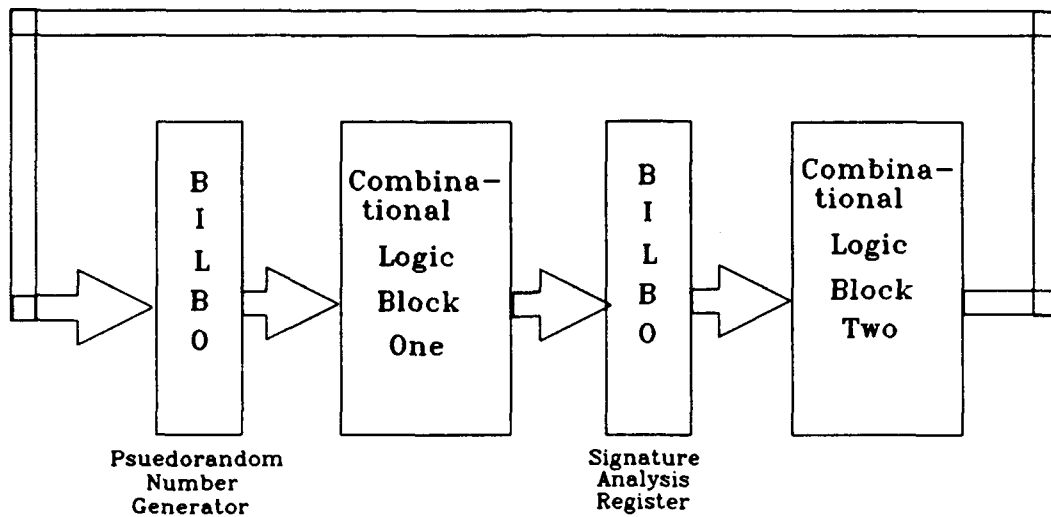
Figure 2.5   Configuration to Test Combinational
Logic Block One   After Ref. 2
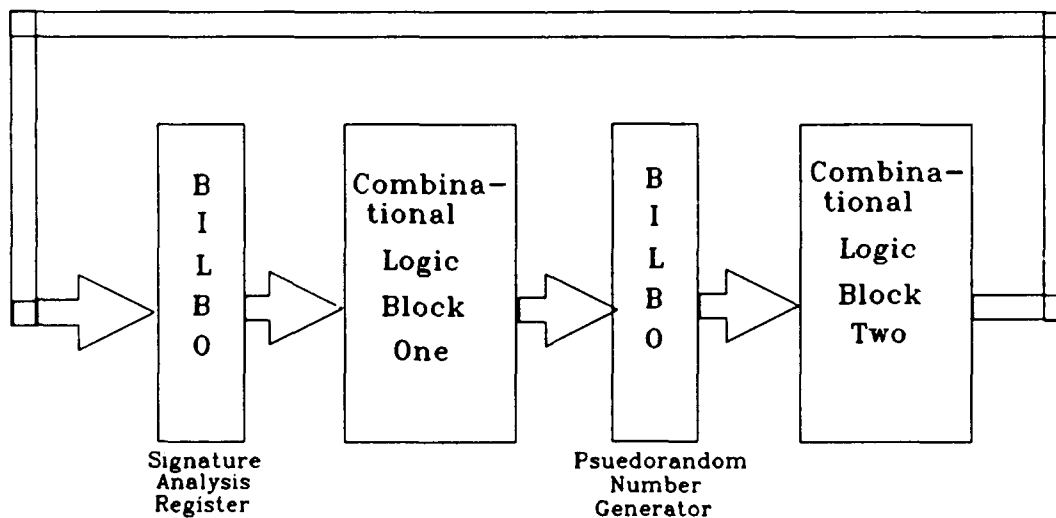
Figure 2.6   Configuration to Test Combinational
Logic Block Two   After Ref. 2

30

BIT technique, that need to be scanned in to achieve a desired fault coverage level [Ref. 7:p. 93].

b. Scan Path Disadvantages

The disadvantages of the scan path technique are related to both external test gear complexity and cost and to the time needed to perform adequate testing. Since all test vector inputs and test result outputs are generated or examined external to the chip, a complex, costly test gear setup is normally needed [Ref. 7:p. 94]. This test gear must be able to both generate and apply the customized test vectors required and then analyze the test results for each test vector input used. Thus, the test gear must work with a much larger volume of data than if the BIT technique is used.

The need to both load each test vector input and extract each test result output in a serial manner can involve a significant overhead of time. Desired testing may take a significantly large number of clock cycles and the testing time overhead will increase with the length of the scan path used. Also, the time overhead involved to utilize a multiple number of test vector patterns is much greater than if BIT, which needs only a single shift-in and shift-out of data for a set of multiple test vector patterns, is used.

c. Built-in Test Advantages

A main advantage of the BIT technique is that it allows a portion of the test functions to be moved from off to on chip. This may make the use of simplified external testing

processes or test gear possible. By generating test vector patterns and compacting test results internally, BIT can greatly reduce the volume of data which needs to be shifted in or out of the chip [Ref. 2:p. 108]. Finally, by combining this lowered overhead with an ability to generate and apply the internally generated test vector patterns at the rated speed of the chip BIT can greatly reduce the total time spent in applying a specific number of test vector patterns to the chip [Ref. 11:p. 379].

d. Built-in Test Disadvantages

One potential disadvantage of using BIT is encountered when either signature or BILBO registers are used to accomplish test result compaction. This problem is termed **aliasing**. Aliasing refers to the situation where the encoded result produced in the signature analysis or BILBO register for a faulty circuit is the same as that produced for a fault free circuit. This occurs when the test data streams for a faulty circuit and fault free both compact to the identical encoded result. The probability of aliasing occurring is a function of the number of bits compacted from the test data stream versus the number of stages in the register. This probability will rise if either the number of bits to be compacted is increased or if the number of stages in the register is decreased.[Ref. 8:p. 106]

A second major disadvantage for BIT centers on the test vector patterns which can be generated by a LFSR network.

Although a LFSR maximum length sequence generator can produce a nearly exhaustive, 2^n-1 set of test patterns for an n-stage LFSR, the sequence order of the test patterns produced may be nonoptimal for achieving a maximum level of fault coverage [Ref. 11:p. 379]. It is possible that there may be no initialization pattern which can be applied to the LFSR to produce a properly sequenced set of test patterns to maximize the fault coverage.

A final disadvantage is that BIT requires additional circuit logic above that needed for the scan path technique. A multiplexer is needed to control the flow of data into the shift register chain and XOR gates are needed both in the feedback path and for the any entry points of test result data which is going to be compacted. The cost of this additional circuit overhead can be looked at in terms of the ratio of circuitry needed for testing to that of the circuitry needed for performance of the chip's required functions.

C. GENESIL IMPLEMENTATION OF THE SCAN PATH TECHNIQUE

The device used during this thesis to investigate the incorporation of DFT into chip design is a 16-bit correlator. It was chosen because it involves an easily understood functional design which can readily incorporate DFT. Additionally, this design was chosen because consideration of DFT issues for it had already been looked at in previous thesis work by Davidson [Ref. 7]. The goal established for the chip design which was fabricated was to incorporate that DFT

technique which would maximize fault coverage for a minimum number of applied test vector patterns. Based on the results and conclusions in Reference 7 about this correlator design, the scan path technique was chosen as the method to use to accomplish the chip design goal.

Implementation of the scan path technique for a Genesil produced design is accomplished by including Genesil Testability Latch Blocks. These blocks consist of a set of serially connected, individual **testability latches**. Each testability latch operates on a single bit of data. By stringing the testability latches together serially a scan path shift register channel is produced. The Testability Latch Blocks have the ability to perform five different operations:

1. During normal (nontest) operation of the chip they serve as data storage latches.

2. The **shift** operation provides the ability to serially load or extract a test vector via the scan path. Each shift operation moves the scan path shift register data one bit further along the scan path in the direction towards the scan path output.

3. The **force** operation is used together with the scan-in process. It provides the ability to take a test vector which was serially loaded into the scan path shift register and force a parallel load of its values into the data storage latches.

3. The **sample** operation is used together with the scan-out process. It causes the values in the data storage latches to be loaded in parallel into the shift register. Values can then be serial shifted out of the chip.

4. The **swap** operation makes possible coupled force and sample operations. The result is that sampled data can be removed from the data storage latches and shifted out at the same time that a new test vectors is shifted in.

34

Through these five operations Testability Latch Blocks are able to enhance controllability and observability within a chip design.[Ref. 12:p. 15-2]

The circuit configuration of a single testability latch is shown in Figure 2.7. Five control gates and three D type transparent latches are used in the testability latch circuit. The control gate signals LOAD, A, B, F and S both enable the gates so as to route data between the latches and provide the signals which cause the latches to be loaded. Latch D serves as the data storage node and is loaded when either the LOAD or F signal goes high. Latches S1 and S2 form the shift register. Latch S1 is loaded when either the A or S signal goes high. Latch S2 is loaded when the B signal goes high. To form a complete scan path the TOUT signal from each testability latch is connected to the TIN signal of the testability latch for the next-most-significant bit. Only the TIN signal for the least-significant bit (LSB) and the TOUT signal for the most-significant bit (MSB) cf the scan path need to be connected to external pins on the chip.[Ref 12:pp. 15-3, 15-4]

The functions performed by the testability latches are based on the control gate signal sequences introduced to the circuit. To load the data storage latch with the value present on the DIN signal line the LOAD signal must be raised high. The latched data value then becomes available on the DOUT signal line. To shift data upwards one significant bit in the scan path shift register the control A and B signals

35
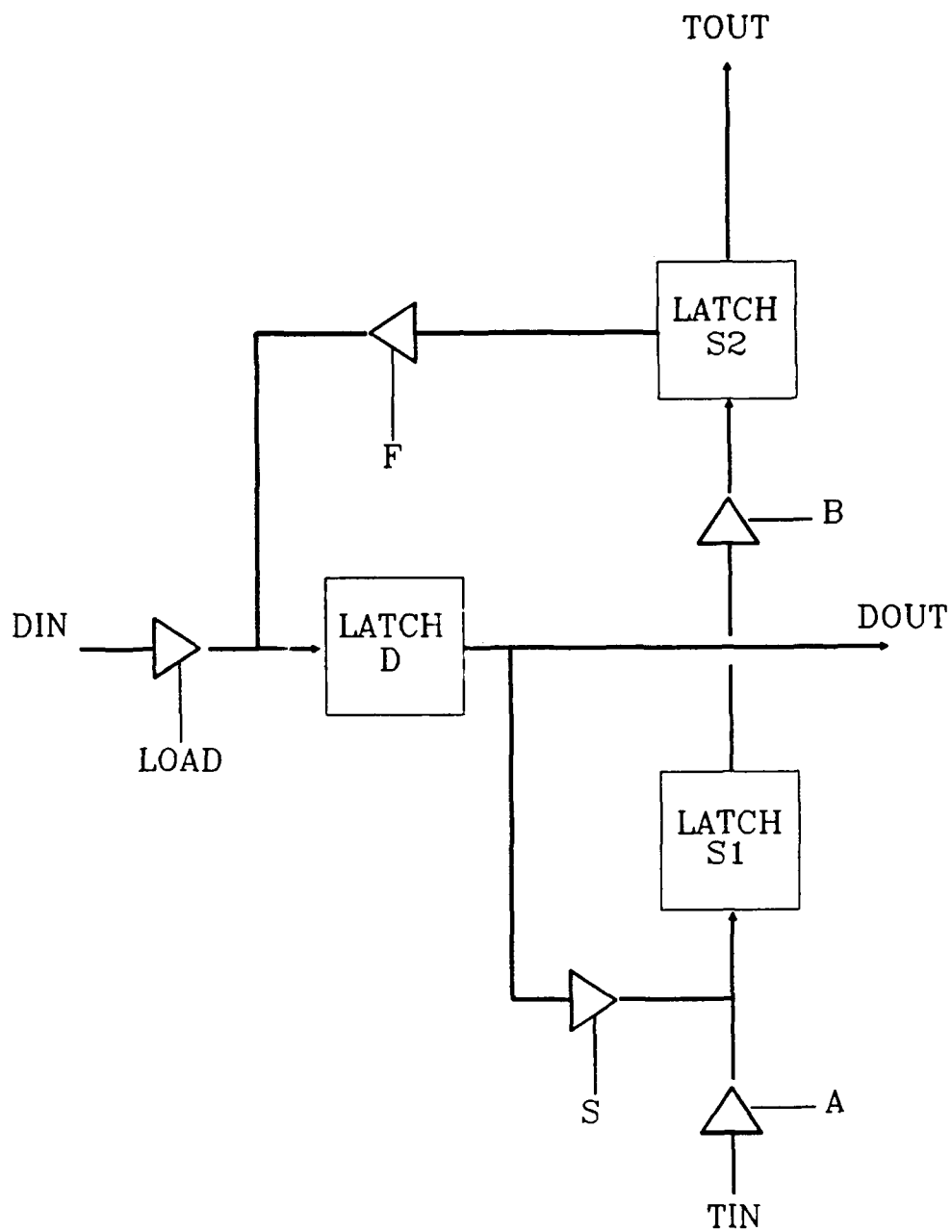
Figure 2.7 Testability Latch Circuit Configuration
After Ref. 12

must be sequentially strobed. To force data to be copied from the scan path shift register to the data storage latch the control F signal must be strobed. To sample or copy data from the data storage latch to the shift register the control S and B signals must be sequentially strobed. To swap or exchange data between the shift register and the data storage latch the control S, F, and B signals must all be sequentially strobed. [Ref. 12:p. 15-5]

In Genesil, using random logic blocks, there are three configurations of basic testability latches available. Their differences are based on the methods used to provide clock and control signals to the testability latches. These three configurations are:

1. The **TLATCHU** configuration. This configuration uses unclocked A, B, F and S control signals which function independently of any clock signals found on the chip. It requires that externally generated control strobes be used to drive the A, B, F and S control signal lines directly. Typically these control strobes originate from off chip. The chip designer must ensure that the timing sequence requirements for the control signals are met to perform the desired operations. Table I speci-fies the different operation modes of the TLATCHU configuration.[Ref. 12:p. 15-6]

2. The **TLATCHG** configuration. It makes use of the phase_a and phase_b clock signals derived from the two-phase global system clock to implement timing relationships used in producing properly timed control sequence orders for the A, B, F and S control signals. Input control signals named M1 and M2 are used to encode the operation which is desired for the testability latch. These signals are latched with additional circuitry and are then decoded to produce the correct A, B, F and S signals. Therefore, only the M1, M2 and LOAD signals need to be generated external to the testability latches (normally from off chip). Table II defines the encoding for and operation modes of the TLATCHG configuration. [Ref. 12:pp. 15-14, 15-15]

37

TABLE I
TLATCHU CONFIGURATION OPERATION

Required Inputs A B F S	Operation Mode
0 0 0 0	NORMAL
1 0 0 0 0 1 0 0	SHIFT*
0 0 1 0	FORCE
0 0 0 1 0 1 0 0	SAMPLE*
0 0 0 1 0 0 1 0 0 1 0 0	SWAP*

*Mode requires a sequence of non-overlapping strobes in the order shown.

TABLE II
TLATCHG CONFIGURATION OPERATION

Encoded Inputs M1 M2		Operation Mode	Decoded Outputs Relative to Clock Phases phase_a phase_b	
0	0	SHIFT	A	B
1	0	FORCE[1]	F	B
0	1	SAMPLE	S	B
1	1	SWAP[2]	S	–

[1]LOAD signal disabled during phase_a of FORCE cycles.

[2]SWAP must be followed immediately by a FORCE cycle to to override Latch D and advance the Latch D contents into Latch S2.

3. The **TLATCHL** configuration. It is similar to the TLATCHG
    configuration except that it uses its own local clock
    vice the global system clock to produce timing informa-
    tion. Use of this local clock allows this configuration
    to perform operations independently of the status of the
    global system clock. The clock signals phase_ta and
    phase_tb are derived from this local clock and provide
    properly timed control sequence orders for the A, B, F,
    and S control signals. Normally the local clock signal
    originates as an additional external input to the chip.
    The M1 and M2 control signals perform in the same manner
    as they do for the TLATCHG configuration. Table III
    defines the encoding for and operation modes of the
    TLATCHL configuration. [Ref. 12:pp. 15-10, 15-11]

As is discussed in Chapter 3, the TLATCHL configuration was

chosen as the means of incorporating a scan path into the

16-bit correlator chip which was fabricated.

TABLE III
TLATCHL CONFIGURATION OPERATION

Encoded Inputs M1 M2	Operation Mode	Decoded Outputs Relative to Clock Phases phase_ta	phase_tb
0 0	SHIFT	A	B
1 0	FORCE[1]	F	B
0 1	SAMPLE	S	B
1 1	SWAP[2]	S	—

[1] LOAD signal disabled during phase_ta of FORCE cycles.

[2] SWAP must be followed immediately by a FORCE cycle to
    to override Latch D and advance the Latch D contents
    into Latch S2.

39

III. DESIGN FOR TESTABILITY IMPLEMENTATION METHODOLOGY

This chapter discusses the methodology used with the Genesil Silicon Compiler to implement a chip incorporating DFT. The different menu features needed, used, and available within Genesil serve as the chapter subdivisions for presenting information on the different actions and multiple criteria considered during the production of a DFT chip design. Although the chapter was broken into subsections based on working with features within Genesil, it must be realized that chip design in Genesil involves a continuous interrelationship between all Genesil features available. Results obtained or decisions made during the progression of design steps while using one Genesil feature will carry over into the results obtained or approach taken while using another Genesil feature. Design in Genesil can be an iterative process involving multiple cycles of design changes due to information obtained from the different Genesil features. Specific information on all aspects of Genesil is provided in the documentation for the Genesil Silicon Compiler. Additionally, Reference 13 provides excellent information and a tutorial on the methodology for and makeup of a Genesil produced design.

To present a high level view of the chip as a whole, this chapter starts by presenting a functional description of the complete correlator chip design. The remainder of the chapter

concentrates on the specific reasoning which was used, the engineering design decisions which were made, and the steps which were taken to produce the chip design. The second section in this chapter discusses the operational and engineering design criteria which were established and the specific methodology steps used within Genesil to produce a chip which met these criteria. The third section emphasizes both the steps used and information gained from functional test simulation and timing analysis. The last section provides information on how Genesil's Automatic Test Generation feature was used both to produce a maximum fault coverage test vector set and to fault grade the test vectors produced during functional test simulation. Although placed last in the chapter, this section also provides further analysis on the process used to decide where to place and how to utilize the DFT technique of a scan path.

A. FUNCTIONAL DESCRIPTION

The chip which was fabricated is a 16-bit correlator. It performs a comparison between an incoming 16-bit word of data, which is placed into the data register, and a preloaded 16-bit word of data found in the reference register. The chip performs a bit by bit comparison between the two registers and produces a 5-bit binary number which indicates the number of bits, from zero to 16, that are positively correlated (matched) between the two registers. Additionally, a 16-bit mask register exists to enable or disable the correlation

41

process for a given set of bits in the data and reference registers. If the mask register has a specific bit set to a logic 1 the correlation results for corresponding bits in the data and reference registers are included in the correlation process. If the mask register has a specific bit set to a logic 0 the correlation results for the corresponding bits in the data and reference register are not included in the final result which indicates the total number of bits that matched. The data, reference, and mask registers are all provided with a means to be selectively loaded in either a serial or parallel manner. The final correlator chip design is partitioned into five main sections: input registers, XNOR register, combiners/testability latches, adder and output. Figure 3.1 is a block diagram showing the relationships and main signals of these sections.

1. Input Registers

The input section of the chip consists of three identically designed input register general modules: **data_in** (the data register general module), **ref_in** (the reference register general module) and **mask_in** (the mask register general module). Each of these input register general modules holds a 16-bit word to be used during the correlation process. These 16-bit words can be controllably loaded in either a serial manner via the **SERIAL_IN** input pin and **serial_in** signal or in a parallel manner using the **IN_PADS[15:0]** input pins and **par[15:0]** signals. The value of the **sp_con** signal, as input

42
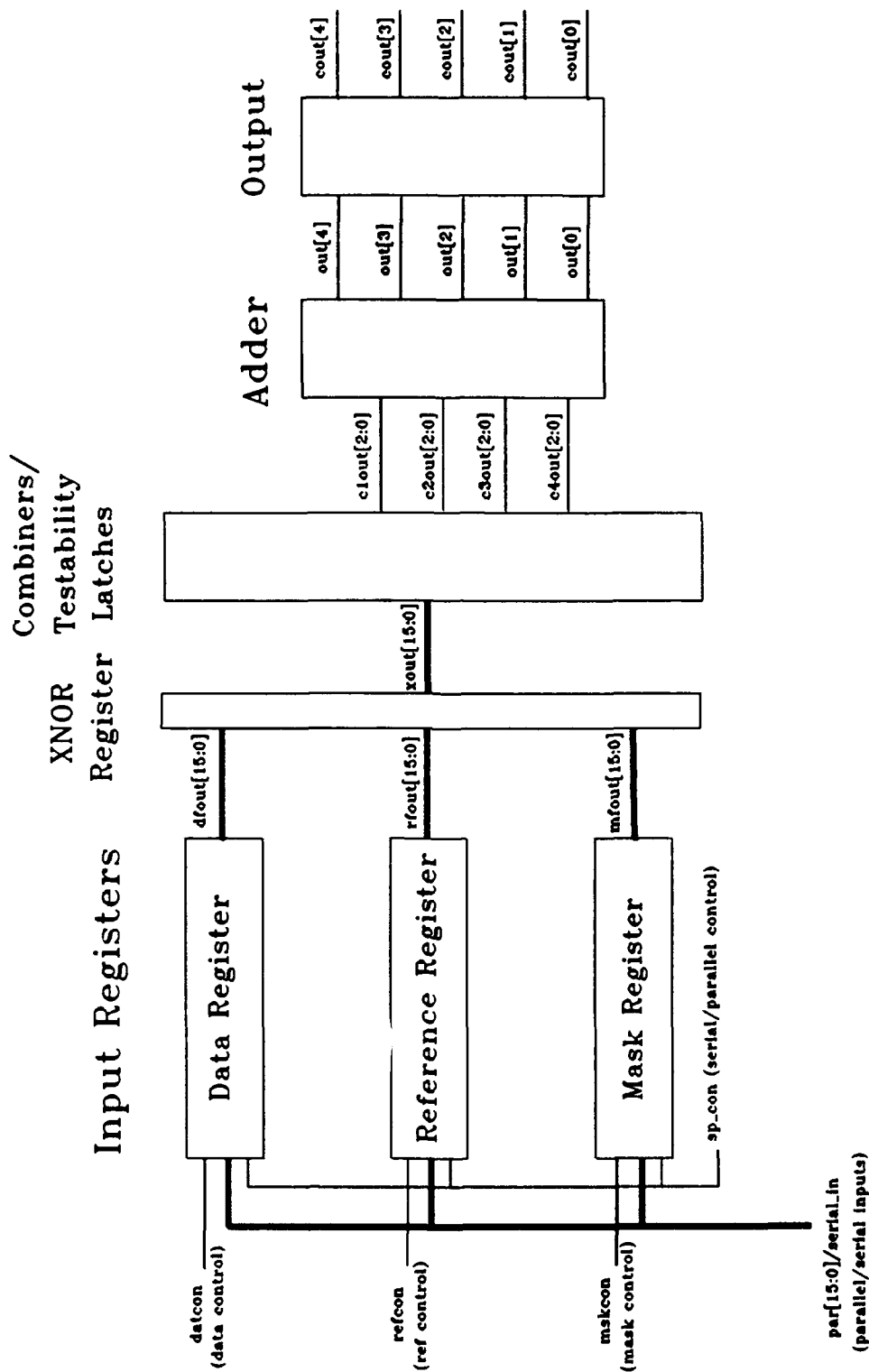
Figure 3.1 Correlator Chip Block Diagram

to the **SPCON** input pin, determines which load method is used. If sp_con is a logic 1 the 16-bit words are loaded in a serial manner. If sp_con is a logic 0 the 16-bit words are loaded in a parallel manner. A load control signal (**datcon** for the data_in general module, **refcon** for the ref_in general module, and **mskcon** for the mask_in general module) is used to control whether an input register general module is allowed to load data which is present at the SERIAL_IN or IN_PADS[15:0] pins or whether loading is disabled for a particular input register general module. As a result, the three input register general modules can controllably load either the same data all at once or individually load different data one at a time via manipulation of their load control signals.

Each input register general module consists of two serially connected 8-bit shift register modules, to hold the 16-bit words, and a control block. The shift register modules (**data1** and **data2** for the data_in general module, **ref1** and **ref2** for the reference general module, and **mask1** and **mask2** for the mask general module) are formed from eight D flip-flop/multiplexer combinations. During serial loads the input for each D flip-flop is the output from the previous D flip-flop/multiplexer stage. The output line for the MSB of the first shift register module is connected to the serial input line of the multiplexer for the LSB of the second shift register module. Thus the two shift register modules are effectively connected in a serial manner to form a 16-bit shift register. For

44

parallel loads, the multiplexers get the input values to pass to the D flip-flops via the par[15:0] signal lines. The sp_con signal controls the multiplexers to determine whether they pass values to the D flip-flops from their serial or parallel load lines. The control block for each input register general module consists of a simple two-input AND gate which has phase_b of the global system clock as one input, the appropriate load control signal as the second input, and which produces the clock signal for the D flip-flops of the shift register modules as its output. Therefore, making the load control signals a logic 1 allows passage of the phase_b clock signal so that the D flip-flops can be loaded with new values. A load control signal set to logic 0 inhibits the clock signal to the D flip-flops thus causing them to retain their present values regardless of any changes to the values present at the chip's serial or parallel input load pins. Figure 3.2 illustrates a 1-bit wide slice from an input register general module.

2. XNOR Register

The XNOR register is a random logic module, labeled **xnorreg**, consisting of 16 two-input XNOR gate, two-input AND gate combinations. One combination is used for each of the 16 bit positions for the correlation process. Figure 3.3 depicts a 1-bit wide slice of the XNOR register module. The two inputs to the XNOR gates are the outputs from the D flip-flops of the data and reference registers. The XNOR gate produces

Figure 3.2   1-Bit Wide Slice of an Input Register

Figure 3.3   1-Bit Wide Slice of the XNOR Register

46

an output which is a logic 1 if these values are positively correlated (match) and a logic 0 if they are different. Thus it in effect compares each corresponding bit position in the data and reference registers. The output of the XNOR gate along with the output of the D flip-flop from the corresponding bit position of the mask register serve as inputs to the AND gate. Therefore, the mask register bits determine whether positively correlated results obtained from the data and reference registers are passed onwards to be counted or are blocked from being considered. A logic 1 in a mask register bit position produces a logic 1 output from the XNOR register in that bit if the data and reference register words match and a logic 0 if they do not. A logic 0 in a mask register bit position causes the XNOR register to output a logic 0 for that bit position regardless of how the data and reference register bits correlate. It thus causes that bit position to become masked from inclusion in the final chip output which indicates the number of positively correlated bits.

3. Combiners/Testability Latches

The purpose of the two combiners, **combiner1** and **combiner2**, is to collectively reduce the output of the XNOR register to four 3-bit binary numbers. Each 3-bit binary number is the summed total of positively correlated, nonmasked bits from a 4-bit wide slice of the data and reference registers as output by the XNOR register. Figure 3.4 shows the logic gate representation of how one 4-bit wide slice of

c1out[2:0]

OR1

OR0

XOR5

XOR4

XOR3

XOR2

AND5

AND4

AND3

AND2

AND1

XOR1

AND0

XOR0

xout[3:0]
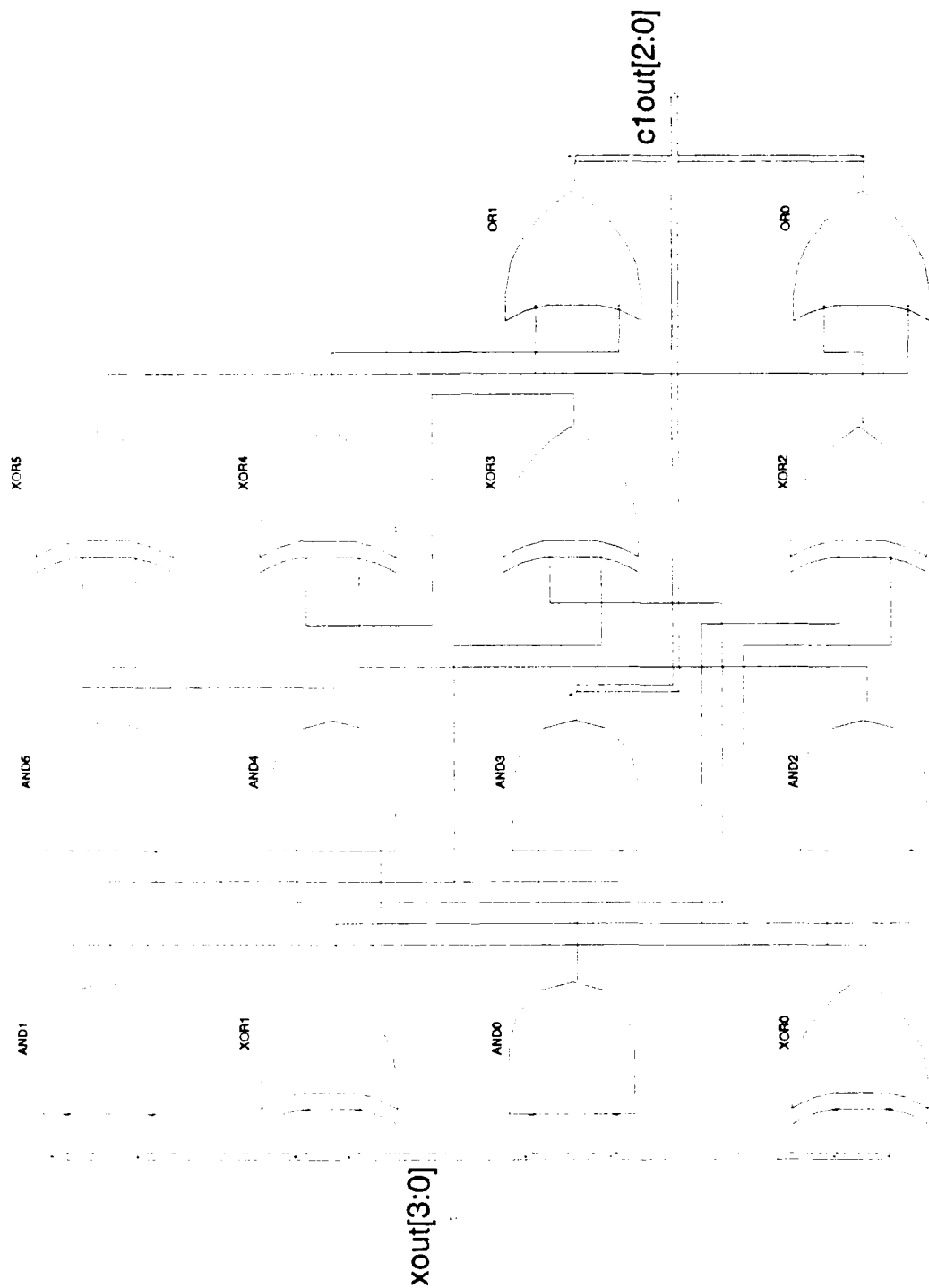
Figure 3.4   Combiner Circuit 4-bit Wide Slice

48

results from the XNOR register output is summed to produce a 3-bit binary number representing the total of positively correlated bits. To incorporate design for testability features into the chip, a scan path consisting of serially connected testability latches was placed between the gates of the combioer function as indicated by the dashed line of Figure 3.5. This dashed line represents the demarcation between the gates, for a given 4-bit slice of the combiner function, which belong to combiner1 and those found in combiner2. A discussion on why the scan path was placed here is provided in the last section of this chapter.

The testability latches themselves are located in the **tlatch32** module which is formed from two 16-bit testability latches (the maximum random logic testability latch size allowed in Genesil) **tlatch0** and **tlatch1**. The testability latches are operationally controlled via the control signals **m1**, **m2**, and **load** which originate from the **M1**, **M2**, and **LOAD** input pins respectively. Additionally, data can be scanned into and out of the chip through the scan path testability latches using the **testin** and **testout** signals from the **TESTIN** input pin and **SHIFTOUT** output pins. The clocking needed for the different operations of the testability latches is provided by the **phase_ta** and **phase_tb** clock signals derived from the clock signal input to the **CLOCK2** input pin.
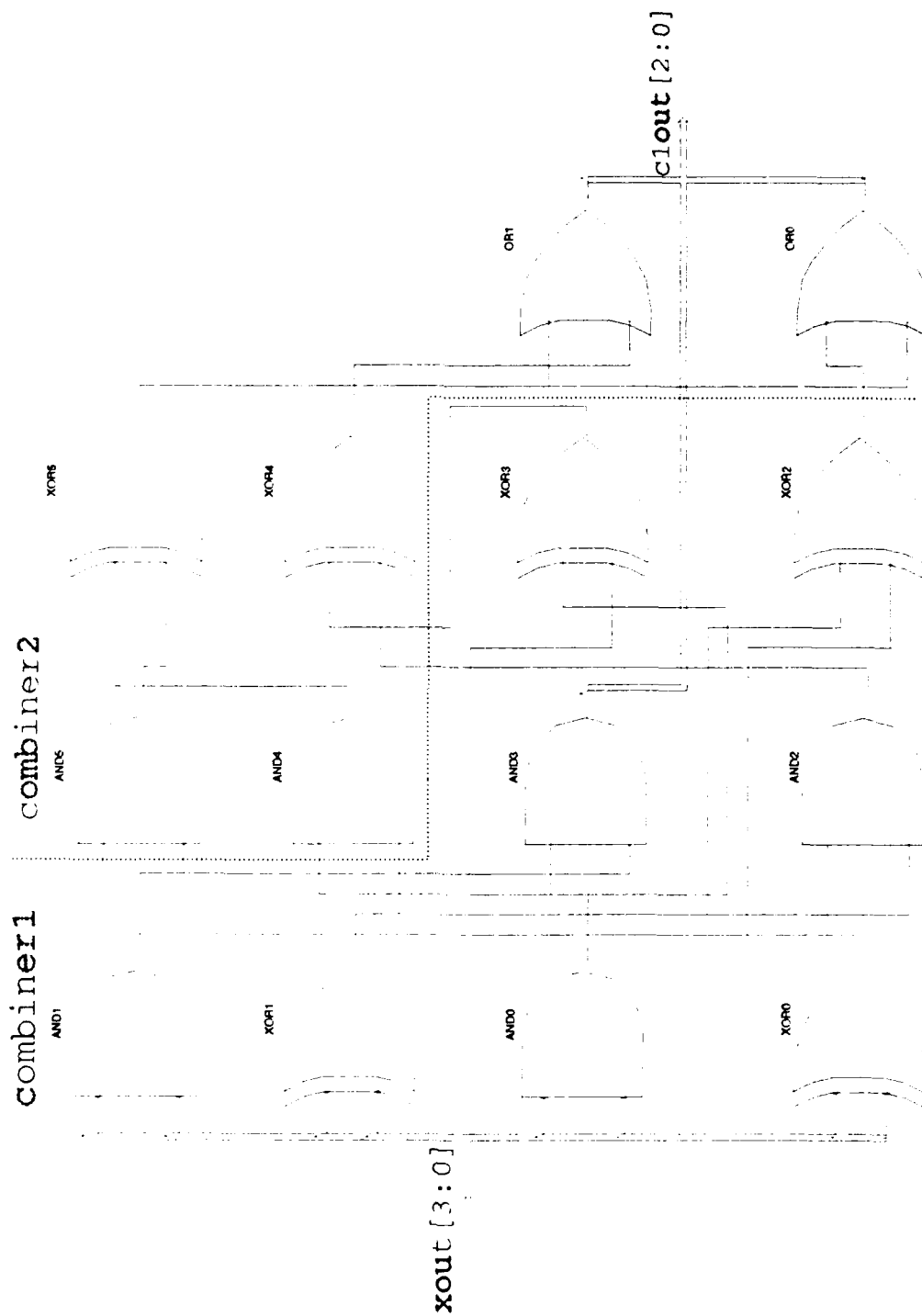
Figure 3.5   Testability Latch Placement

4. Adder

The purpose of the **adder** section is to take the four sets of 3-bit binary numbers produced as an output of the combiner function and add them together to produce a 5-bit binary number which represents the sum total of all positively correlated, nonmasked bits between the 16-bit words in the data and reference registers. The adder module consists of three adders. There are two 3-bit adders, **ADDER0** and **ADDER1**, which each take two 3-bit binary numbers output from the combiner function and add them to produce a single 4-bit number. The third adder, **ADDER2**, takes the two 4-bit numbers produced by ADDER0 and ADDER1, adds them together, and produces the 5-bit binary number result on the signal lines **out[4:0]**. Figure 3.6 shows the configuration of the adder module.

5. Output

The **output** module is included on the correlator chip as a means whereby the output results of the correlation process can be controllably turned on or off. This section consists of five two-input AND gates as shown in Figure 3.7. Each AND gate has as one of its inputs a single bit from the final binary number result produced by the adder section and as its other input the control signal **output** which originates from the **OUTCON** input pin of the chip. The outputs from the AND gates form the final output signals **cout[4:0]** which are sent out of the chip via the output pins **OUT_PADS[4:0]**. Thus,
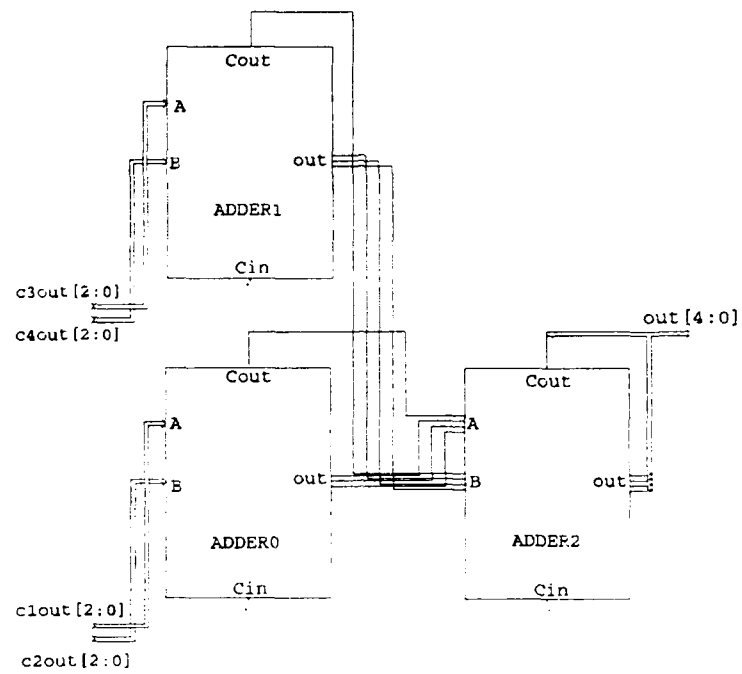
Figure 3.6 Adder Section Configuration
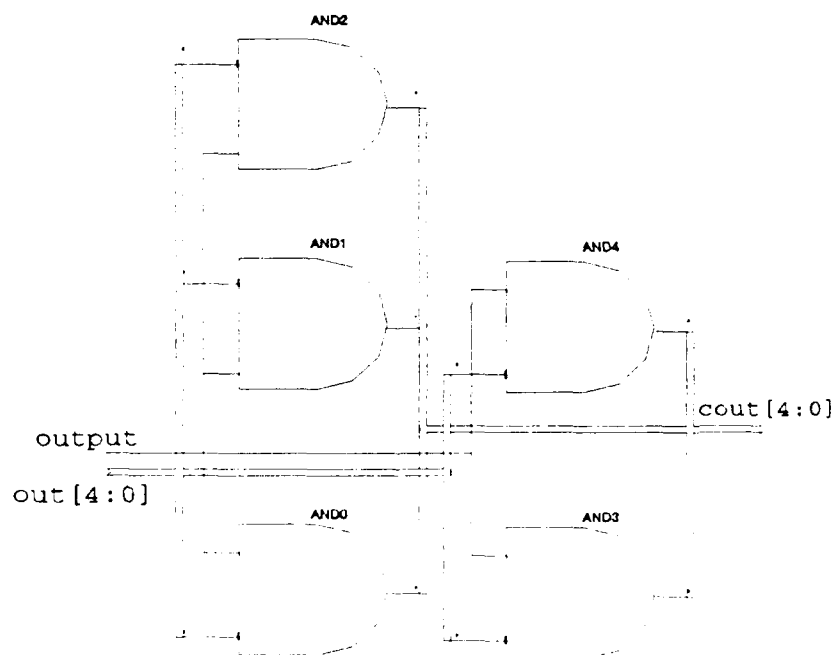
Figure 3.7 Output Module Configuration

the control signal named output can either activate the chip allowing results to be presented to the outside world or inhibit the presentation of these results.

B. DESIGN CRITERIA, DECISIONS AND TECHNIQUES

The final 16-bit correlator chip design produced was affected by both specific engineering design and operational criteria and by the procedural steps necessary for designing a chip in Genesil. Since Genesil produces chips by utilizing a library of already designed objects (Genesil blocks), the chip designer must accept the limitations of producing designs done at the gate level or above instead of working at the transistor level. Therefore, the cost of working in Genesil is less ability to optimize the design in terms of performance and chip size. In contrast, using Genesil provides benefits related to areas such as reduced design time, a lower level of required designer knowledge, easy to use simulation capabilities and a fast means of determining fault coverage levels.

For comparison purposes, two 16-bit correlator chips were designed in Genesil. **BASIC_CHIP** is the 16-bit correlator design discussed in the fu :tional description <u>without</u> any DFT technique included. **DFT_CHIP** is the same design but with the addition of testability latches to form a scan path as discussed in the functional description. Within the limitations of Genesil, the following set of criteria was established for the DFT_CHIP:

1. Functionally, the DFT_CHIP should produce the same correlation results for a given set of bits in the data, reference, and mask registers as does the BASIC_CHIP.

2. The only difference between the DFT_CHIP and the BASIC_CHIP should be the addition of testability latches, along with pins for the supporting control, clock, and I/O signals, to form a scan path.

3. The design area size for the DFT_CHIP should be minimized to the greatest extent possible to lower fabrication costs.

4. The fabrication line and feature size chosen for usage in Genesil should be compatible with those possible for chip fabrication via MOSIS.

5. There should be a maximum of 40 pins for the DFT_CHIP to allow it to be accommodated in the test jig available for use with the commercial tester.

6. The testability latches used should be of the TLATCHL type to gain the advantages of using a local clock to perform operations in the scan path.

7. Fault coverage for the chip should be maximized through the choice of where to place the scan path to increase observability and controllability.

8. The DFT_CHIP should be designed to utilize the highest clock frequencies possible within the limitations of the other criteria.

As can be seen, these criteria are divided between those related to fabrication requirements and those related to desired performance or operational characteristics.

The overriding criteria for the fabrication requirements became the need to layout a chip design that was both small enough and oriented properly to fit within a maximum design area size of 4.6mm wide by 6.8mm high. This size limitation originated from the need to keep the fabrication cost for the chip within a reasonable limit. The size of 4.6mm by 6.8mm is the largest which could be used for fabrication via MOSIS

within the budget allocated for the chip. The criterion emphasized for performance was maximizing the fault coverage possible for the chip. This necessitated intelligent placement of the scan path to enhance observability and controllability. Information on how this was accomplished is located in the subsection of this chapter titled Automatic Test Generation and Fault Coverage.

1. Design Decisions and Techniques to Minimize Size

The design steps for the DFT_CHIP followed the normal sequence used in Genesil. First, blocks were chosen from the available library and combined to form modules. Based on utilizing individual gates for the designs of the input registers, XNOR register, combiner and output sections of the chip, Genesil **random logic blocks** were use td accomplish this. Netlisting was used to specify the signal interconnections between the blocks. The objects making up the input registers were further netlisted and then floorplanned together to form larger general modules. Note that at this point the input register general modules consisted of a single 16-bit shift register and the control block. These actions resulted in the following modules and general modules being created: data_in, mask_in, ref_in, xnorreg, combiner1, combiner2, tlatch32, adder and output. Next, modules and general modules were placed into the chip and the **independent blocks** (pads) for the input, output, clock, and power supply pins were added. The chip as a whole was then floorplanned.

Finally, the chip was compiled using the VTI-CN20A 2.0um n-well CMOS process from VLSI Technology, Inc. as the choice from among the Genesil supported fabrication lines. To determine the size of the chip designed, a route plot as shown in Figure 3.8 was produced.

Since the first design attempt produced a chip of size 5.902mm by 8.276mm, chip size reduction was mandated. Most of the work done to reduce chip size involved changes to the way the chip was floorplanned. Floorplanning in Genesil consists of three categories: placement, pinout and fusion. **Placement** determines the manner and orientation in which objects are physically located next to each other. For the initial design attempt the Genesil **AUTO_PLACEMENT** (automatic placement) feature, **AR_PLACE** option was utilized. This feature and option performs placement based on an attractive repulsive algorithm which orients objects with many common signals next to each other irrespective of their sizes [Ref. 14:p. 3-7]. Figure 3.9 shows the placement configuration which resulted from using automatic placement. To optimize the size in the subsequent design attempts, manual vice automatic placement of the objects was used. Manual placement allows a designer to specify exactly where items should be located in relationship to each other. For the DFT_CHIP this allowed the input register general modules to be oriented along the height axis, at right angles to the other modules, to attempt to form a rectangle smaller than the 4.6mm by 6.8mm space allotted.

56
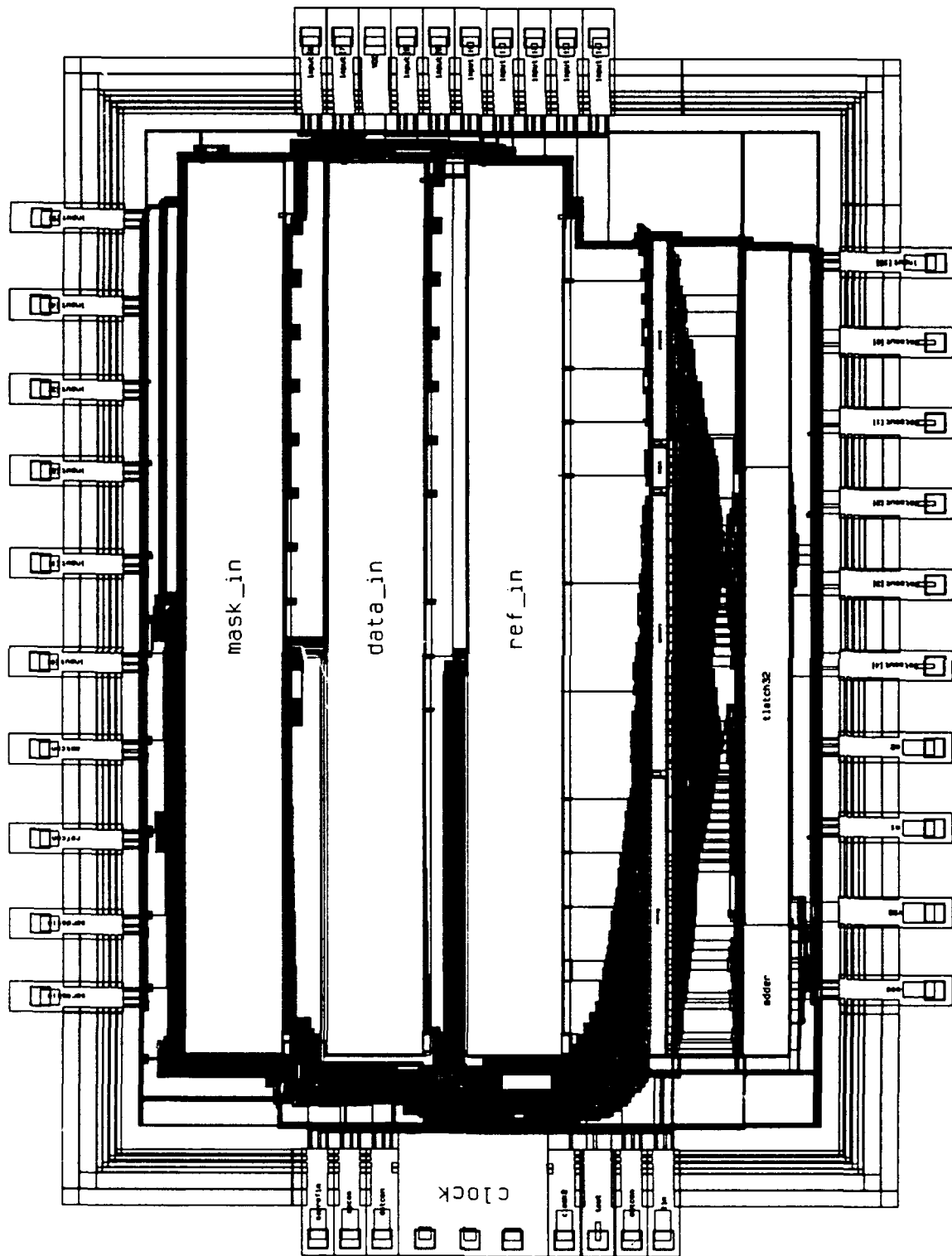
Figure 3.8   Route Plot for First Design Attempt
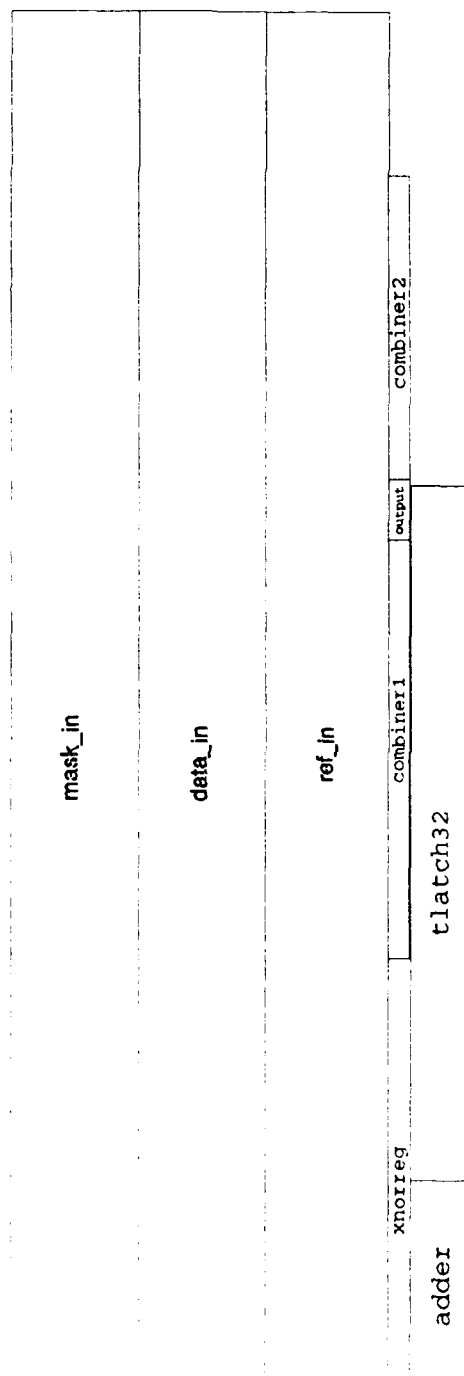
Figure 3.9   Placement Configuration from using
AUTO_PLACEMENT Feature, AR_PLACE Option

Figure 3.10 shows the modified placement configuration which resulted from the use of manual placement.

Manual placement still did not solve the size problem. The chip produced was almost within the width tolerance of 4.6mm but was elongated along the height axis due to the thin, long nature of the input registers. To solve this problem the input registers were redesigned. The 16-bit shift register sections were broken into the two 8-bit shift register sections discussed in the functional description. The input register general modules were then refloorplanned, using manual placement, to place the two 8-bit shift register sections side-by-side. The resulting input register general modules were approximately half as elongated and twice as wide as the original configuration. Figure 3.11 shows the placement configuration of the data_in input register prior to the change. Figure 3.12 (shown at a larger scale) shows the results after the change. This modification turned out to be the single largest size reducing decision made for the chip. Not only did it reduce the elongation of the height axis of the chip but it also caused less nonoccupied/nonutilized space to be present within the chip design.

To further reduce the size dimensions, the other two aspects of floorplanning were considered. First, the floor-planning **pinout** feature was utilized. Within general modules the pinout feature allows choices to be made as to which side of a general module (north, east, south, or west) connectors

59

adder

output | combiner2

tlatch32

combiner1
xnorreg

```
r          d          m

e          a          a

f          t          s

_          a          k

i          _          _

n          i          i

           n          n
```
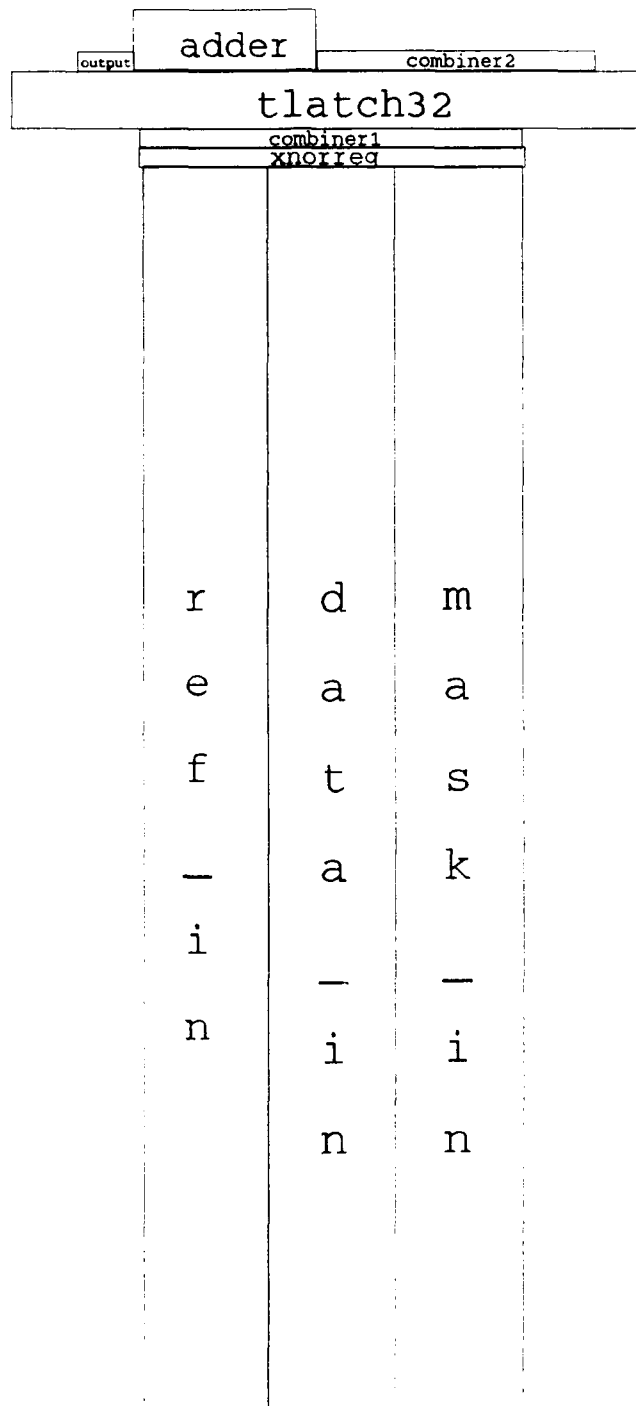
Figure 3.10   Placement Configuration from using
Manual Placement

data

datcon
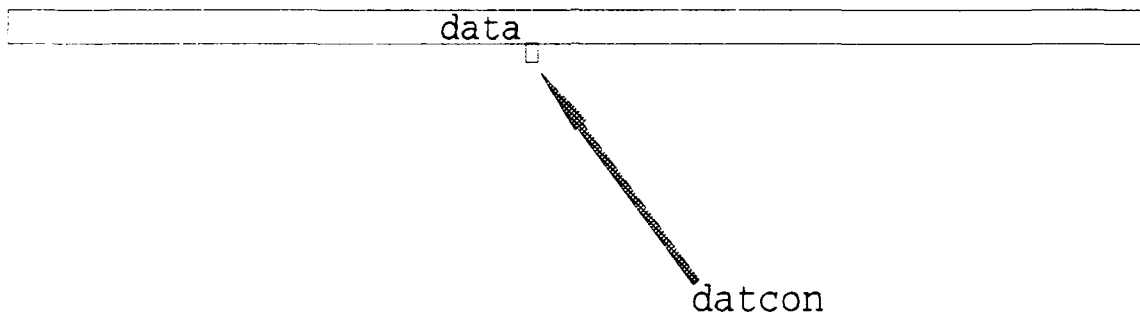
Figure 3.11   Input Register Placement for Single
16-bit Shift Register Section

data1

data2

datccn
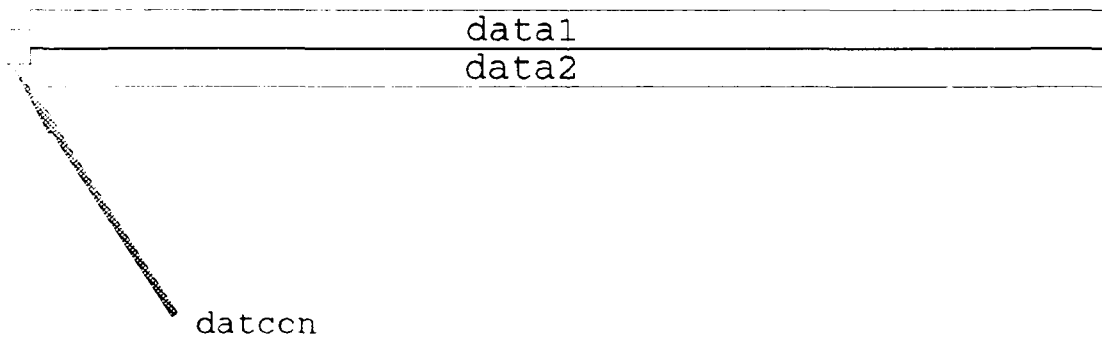
Figure 3.12   Input Register Placement for Two
8-bit Shift Register Sections

for signals going to or from the general module will be placed. By considering how general modules have been placed within the chip a designer can optimize the routing direction of signals between general modules and other chip objects. Connectors for signals in the general modules should be placed on the side of the general module closest to the chip objects which share the signals. Figure 3.13 is an example of the pinout form used to specify connector edge locations for a general module. At chip level pinout determines the pad placement locations for signals which exist external to the chip. By placing pads for specific signals as close as possible to the objects using these signals wire lengths and routing complexity within the chip can be decreased.

The last feature of floorplanning utilized to reduce the chip size was the **fusion** feature. Fusion determines the assignment of routing channels which affect wire routing within the floorplan of the chip [Ref. 15:p. 7.5]. Routing channels are formed through the fusion of two objects, the fusion of one object to a previously defined fusion region, or the fusion together of two previously defined fusion regions [Ref. 15:p. 7.5]. Chip size can be minimized by performing fusion in a sequence that fuses together items whose adjacent boundaries are the same length [Ref. 15:p. 7.5]. Genesil offers the features of both **AUTO_FUSE** (automatic fusion) and **FUSE** (manual fusion). AUTO_FUSE automatically fuses together all items not already fused but does not necessarily produce

```
**********************************************************************
              Genesil Version v7.1 -- Sat May  5 15:06:22 1990
Module: ~genpooler/pooler/DFT_CHIP/data_in
    Floorplan
**********************************************************************

Mode:    ADD_CONNECTOR   MOVE_CONNECTOR   REMOVE_CONNECTOR

Edge:    NORTH   SOUTH   EAST   WEST

NORTH_CONNECTOR   EAST_CONNECTOR      SOUTH_CONNECTOR   WEST_CONNECTOR
  dfout[0]_____  >_____    serial_in_____  >_____
  datcon_____  >_____    sp_con_____ _
  dfout[1]_____                       >_____
  dfout[2]_____
  dfout[3]_____
  dfout[4]_____
  dfout[5]_____
  dfout[6]_____
  dfout[7]_____
  dfout[8]_____
  dfout[9]_____
  dfout[10]_____
  dfout[11]_____
  dfout[12]_____
  dfout[13]_____
  dfout[14]_____
  dfout[15]_____
  par[0]_____
  par[1]_____
  par[2]_____
  par[3]_____
  par[4]_____
  par[5]_____
  par[6]_____
  par[7]_____
  par[8]_____
  par[9]_____
  par[10]_____
  par[11]_____
  par[12]_____
  par[13]_____
  par[14]_____
  par[15]_____
  phase_a_____
  phase_b_____
  >_____
```

Figure 3.13   Pinout Form to Specify Connector Edge Locations

the most efficient floorplan [Ref. 15:p. 7.17]. To help reduce the chip size a change was made from using automatic fusion on the first design attempt to using manual fusion on subsequent design attempts. The ability to reduce chip size through intelligent manual fusion decisions was validated during this process. However, if manual fusion is going to be used it must be done carefully since it was determined that unwise fusion order choices can have highly adverse effects on overall chip size.

Upon completion of all the changes discussed concerning floorplanning the chip size had been reduced to 4.743mm by 6.727mm. To achieve the additional small reduction in size needed for the width axis two additional design decisions were considered. First, Genesil provides the choice of using either TTL or CMOS drivers for output pads. Output pads using CMOS drivers are larger than those using TTL drivers and will increase the size of the pad ring surrounding the core of the chip [Ref. 16:p. 5-4]. Secondly, Genesil provides two options for electro-static discharge (ESD) protection for input pads. ESD circuitry is used to prevent the damaging buildup of large static voltage potentials at input pads, which is common to CMOS chips, by providing a low-impedance discharge path through diodes to either the VDD (power) and or the VSS (ground) supplies [Ref. 16:p. 5-3]. Genesil provides two ESD protection options when specifying the input pad configuration. **NP-PROTECT** uses protection diodes connected to both the

VDD and VSS supplies but produces pads larger in size than **N-PROTECT** which provides only a protection diode to the VSS supply [Ref. 16:p. 5-3]. The initial chip design utilized TTL output drivers and the NP-PROTECT option. To achieve the remaining size reduction needed, the final DFT_CHIP design sacrificed some degree of protection by using only the N-PROTECT option. This choice reduced the final design to a size of 4.550mm by 6.627mm. Figure 3.14 is a route plot for the DFT_CHIP which shows the final configuration results obtained for the overall size minimization process.

2. Additional Design Decisions and Techniques

The remaining design decisions made for the DFT_CHIP can be split into those related to scan path operations and those related to pad specifications. Based on the criteria of desiring scan path operations to be able to proceed independently of the global system clock, the TLATCHL testability latch configuration was chosen to form the elements of the scan path. Only this Testability Latch configuration, from among those available in Genesil, provides the ability to work with a local clock for scan path operations. The criteria requirement itself was established to enhance chip capabilities by providing both a method to perform in-system testing of an operating chip and to allow operation of the scan path a different clock rate from that of the rest of the system. In-system testing for a chip involves checking its operating condition without removing it from its physical working

65
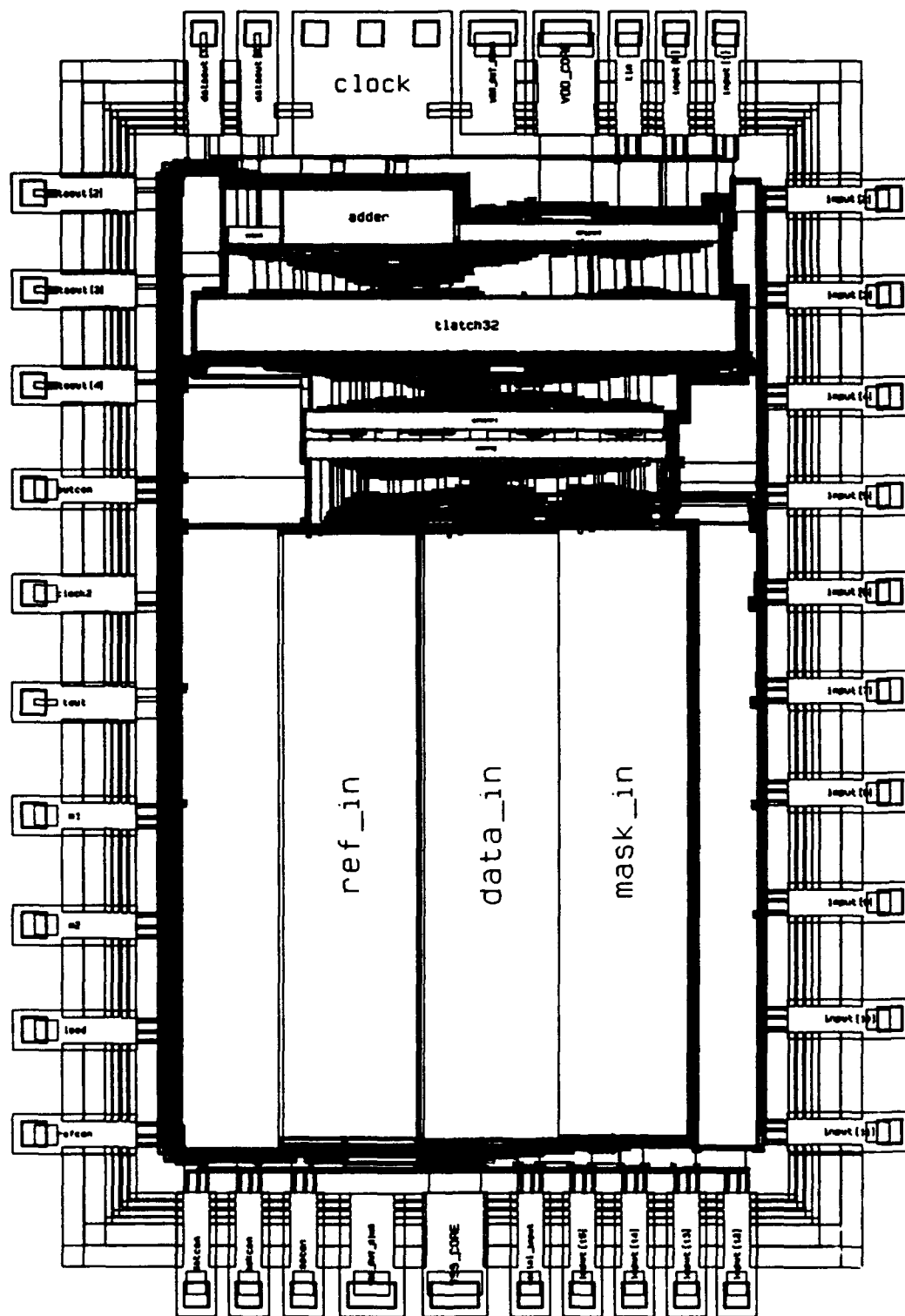
Figure 3.14   DFT_CHIP Final Configuration Route Plot

66

environment. This could be done in the DFT_CHIP by halting

normal chip operations through either disabling the load

control signals (datcon, mskcon and refcon) to the input

registers or through disabling the testability latch load

signal. The chip would then be in a test mode where the local

scan path clock scans test vectors into or out of the chip to

check its operation. The use of the TLATCHL configuration was

also chosen to attempt to make possible the use of a clock for

scan path operations that both runs at a higher speed than the

global system clock and that can continue scan path operations

during periods when the global system clock is not operating.

These features would help speed the rate at which test vectors

or test results could be scanned into or out of the chip and

also would enhance the chip's in-system testing capabilities.

Design choices for the pad specifications were made to

enhance operational performance while still being conservative

enough to help insure that the fabricated chip would operate

satisfactorily. First, pads for power supplies had to be

provided to the chip. Genesil produces designs which require

power for two separate regions: the chip core and the pad ring

[Ref. 16:p. 5-59]. Options exist in Genesil to provide this

power through either separate VDD and VSS pad pairs for each

region or through a single combined pad pair for both regions.

Except for small pad-limited designs, the combined option is

not recommended since it does not provide any isolation from

noise which might be generated by the output pad drivers

[Ref. 16:p. 5-61]. Therefore, the conservative choice of using separate VDD/VSS pad pairs for the riog and core power supplies was made for the DFT_CHIP.

Next, the specifications were made for the clock pads used by the input signals for the global system clock and the scan path local clock. Genesil clock pads convert a single external clock signal into internal two-phase non-overlapping clock pair signals required to operate genesil blocks [Ref. 16:p. 5-15]. The DFT_CHIP uses the clock pad option of **NO_DIVIDE** which provides internal clock signals at the same frequency as the external clock signal. Clock pads have their own power source requirements which depend on the loading placed on them. The primary global system clock must be provided with its own power pads (VDD and VSS) which are isolated from the ring power supplied to the other ring pads [Ref. 16:p. 5-25]. To accomplish this requirement the **LOCAL_ISOLATED** option was chosen for the primary global system clock pad. This choice produced a clock pad with its own VDD and VSS inputs attached. For additional clock pads in a multiple clock design the requirement for isolated power is dependant upon the degree of loading placed on the clock signals. A lightly loaded clock ($\leq 20pF$) can utilize power distributed by the ring power pads vice needing its own power supplies [Ref. 16:p. 5-26]. Since the local clock for the scan path testability latches met this requirement, ring power was used vice choosing isolated power. Note that the maximum

operating frequency to be used for the clocks is not chosen in the pad specification but rather is done during the netlisting process for the clock signals.

Decisions concerning the input pads were made next. The decision on ESD was described previously in the section on minimizing chip size. The other design decision for the input pads was to choose whether external signals would be latched or passed completely untouched from outside to inside the chip. For the DFT_CHIP, Genesil's **Direct** option, which provides a transparent latch at the input pad, was chosen. A latch configuration was chosen to ensure that there would be at least a half clock cycle during which unchanging input data could propagate to and be placed into the input registers. Input data is sampled and passed during phase_b of the latch clock signal and is held during phase_a [Ref. 16:p. 5-39]. After chip fabrication, a major design error was discovered concerning the clock signal used to control the latch for the scan path input pad. Instead of specifying sampling during phase_tb of the local scan path clock, the genesil default of phase_a of the global system clock was accepted as the controlling clock signal. The result of this design oversight is the negation of the ability to conduct scan-in operations for the scan path at either a rate faster then the global system clock or when the global system clock is not operating. Since this ability was part of the reason for choosing the TLATCHL type Testability Latches, this error was quite costly

in terms of available chip operational performance character-
istics.

Lastly, decisions were made concerning the output
pads. Output pads also have options concerning the presence
or absence of latches. Again a transparent latch option was
chosen to help ensure that the final results presented off
chip would be stable for at least half a clock cycle. The
transparent latch for output pads samples values during
phase_a of its controlling clock and holds values during
phase_b [Ref. 16:p. 5-39]. The controlling clock signals were
correctly specified for the output pads. The output pads for
the correlated results have their latches controlled via the
global system clock. The scan path output pad latch uses the
local scan path clock like it should thus allowing scan-out
operations to proceed either at a faster rate then that of the
global system clock or during periods that the global system
clock is not operating. Finally, Genesil offers options on
the drivespeed available for the output pads. The choice
which can be made is dependant upon the number of VDD/VSS
pairs present to provide ring power. A single pair as in the
DFT_CHIP can drive six output pads at very high speed, 12 pads
at high speed, 20 pads at low speed, or 40 pads at very low
speed [Ref. 16:p. 5-37]. Since only six output pads exist for
the DFT_CHIP, the very high speed **DRVSPEED3** option was chosen.

C. SIMULATION AND TIMING ANALYSIS

Genesil's ability to perform simulation and timing analysis provides a convenient means to evaluate and analyze chip performance and functionality characteristics during the design process. Simulation and timing analysis are independent operations within Genesil's working environment. Both of them can and should be used during the entire design process for a chip. By performing analysis on individual modules before combining them into a chip design, timing or logic functionality errors can be easily debugged. Upon completion of a chip design, simulation and timing analysis provide a final verification that the design performs as desired.

1. Simulation

Simulation within Genesil is the process by which a chip design is evaluated to verify that both the implemented design and the physical layout generated from that design function as intended. Through the simulation process, outputs can be checked against a given set or sequence of inputs to verify that the design is logically correct. Genesil performs simulation using either a functional model or a switch-level model.

Simulation done using a functional (GFL) model is a technology and layout independent process. GFL simulation utilizes gate-level, zero-delay models of objects within the design [Ref. 17:p. 2-4]. Technology and layout independence is achieved by considering only the circuit functionality

71

characteristics and changes in input signals rather than looking at the actual delay times found within a circuit design [Ref. 17:p. 1-2]. GFL simulation uses a demand-evaluation algorithm which simulates only the minimum logic necessary to check for correct results [Ref. 17:p. 2-1]. Only block definitions and netlisting need to be accomplished prior to running a GFL simulation test.

In contrast, switch-level (GSL) model simulation is designed to verify functionality at a switch level once a particular technology is specified and the object layout is completed via use of the floorplanning and compilation processes [Ref. 17:p. 2-2]. GSL simulation uses an event-driven algorithm which forces any changes caused by an event to propagate through the design based on detailed timing information obtained from the Genesil Timing Analyzer for the particular technology and layout chosen [Ref. 17:p. 2-2]. GSL simulation is normally used only as a final design verification step during the design process.

Simulation operations in Genesil can be done in either a manual interactive mode or in an automatic control mode. The manual mode requires that the user specify each input, manually advance time via cycling the clocking signals, and individually verify each output. The automatic control mode allows simulation to proceed without user interaction by either operating on a set of test vectors which provide input signals and expected output signals or by using Genesil

72

simulation routines called **check functions**. The automatic control mode both runs faster than the manual mode and if test vectors are used can provide error messages to indicate differences between actual and expected output results [Ref. 17:p. 1-3].

Check functions in Genesil provide an automated approach to both simulate circuit operation and to generate test vector sets. Check functions consist of user defined simulation routines written in a proprietary Genesil language called **GENIE**. The GENIE language provides basic commands which can accomplish all the needed operations to perform simulation on an object. Reference 17 Appendix B provides specifications for the GENIE commands which can be used during simulation. In the manual interactive mode these commands are issued one at a time. By combining these basic commands into a check function routine, a complete simulation task can proceed by using only the check function name as a command. High level check functions can also be written which call lower level check functions and individual GENIE commands. This allows complete simulation procedures to be written for use in a batch type mode.

If check functions and GENIE commands are used in conjunction with the **traceobj** command, test vectors can be produce from simulation runs. The traceobj command causes all inputs and outputs obtained from the simulation run to be captured in a **MASM** test vector file. The MASM test vector

file format contains a list of all the inputs presented and the outputs obtained at each timepoint during the simulation. This MASM format is the same as that used by the Automatic Test Generation process. Therefore, test vectors produced via the simulation process using the traceobj command can be fault graded in the Automatic Test Generation program. Similarly, test vectors produced by the Automatic Test Generation Program can be run in the simulator to verify their correctness. Finally, test vector files for the physical testing of a chip design can be obtained by porting the MASM test vector files to the format needed by a commercial chip tester. Figure 3.15 shows the format of a MASM test vector file.

Simulation results can be presented in either a numeric mode on the message screen or as a screen-based output which can show individual signals in both a tabular and waveform type format. Numeric outputs to the message screen are obtained by using the **RUN_VECTORS** command on a set of simulation test vectors which have already been produced in MASM file format. To produce screen-based outputs a formatted screen must be set up for use as detailed in Chapter 4 of Reference 17. Figure 3.16 and Figure 3.17 are examples of the message and formatted screen type results which can be obtained.

To accomplish simulation of the DFT_CHIP, check functions were written and used to allow simulation steps to progress in an automated manner. Two categories of check

```
CODEFILE
INPUTS
CLOCK(clock),CLOCK2(clock),DATCON(to=0),IN_PADS[15:0](to=0),
LOAD(to=0),M1(to=0),M2(to=0),MSKCON(to=0),OUTCON(to=0),
REFCON(to=0),SERIAL_IN(to=0),SPCON(to=0),TESTIN(to=0);
OUTPUTS OUT_PADS[4:0],SHIFTOUT;
CODING(ROM)
@0   <001000000000000000000100111000 >......;
@5   <011000000000000000000100111000 >......;
@10  <111000000000000000000100111000 >00000.;
@15  <101000000000000000000100111000 >00000.;
@20  <001000000000000000000101111000 >00000.;
@25  <011000000000000000000101111000 >00000.;
@30  <111000000000000000000101111000 >00000.;
@35  <101000000000000000000101111000 >00000.;
@40  <000000000000000000000100010000 >00000.;
@45  <010000000000000000000100010000 >000000;
@50  <110000000000000000001000010000 >000000;
@55  <100000000000000000000100010000 >000000;
@60  <000111111111111111111100110000 >000000;
@65  <010111111111111111111100110000 >000000;
@70  <110111111111111111111100110000 >100000;
@75  <100111111111111111111100110000 >100000;
@80  <000111111111111111111100011000 >100000;
@85  <010111111111111111111100011000 >100000;
@90  <110111111111111111111100011000 >000000;
@95  <100111111111111111111100011000 >000000;
@100 <001000000000000000000100010000 >000000;
@105 <011000000000000000000100010000 >000000;
@110 <111000000000000000000100010000 >000000;
@115 <101000000000000000000100010000 >000000;
@120 <001000000000000000001100010000 >000000;
@125 <011000000000000000001100010000 >000000;
@130 <111000000000000000001100010000 >000010;
@135 <101000000000000000001100010000 >000010;
@140 <001000000000001001010001 0000 >000010;
@145 <011000000000001001010001 0000 >000010;
@150 <111000000000001001010001 0000 >000100;
@155 <101000000000001001010001 0000 >000100;
@160 <001000000010010001010001 0000 >000100;
@165 <011000000010010001010001 0000 >000100;
@170 <111000000010010001010001 0000 >000110;
@175 <101000000010010001010001 0000 >000110;
@180 <001000000010010001110001 0000 >000110;
@185 <011000000010010001110001 0000 >000110;
@190 <111000000010010001110001 0000 >001000;
@195 <101000000010010001110001 0000 >001000;
@200 <001000100100011010010001 0000 >001000;
@205 <011000100100011010010001 0000 >001000;
@210 <111000100100011010010001 0000 >001010;
@215 <101000100100011010010001 0000 >001010;
@220 <001001000110100010110001 0000 >001010;
```

Figure 3.15 MASM Test Vector File Format

75

```
) trace running from vecspara          Sat Jun  2 13:30:54 1990
)      CCD I     LMMMORSS T    O  S
)      LLA N     012SUEEP E    U  H
)      OOT _     A  KTFRC S    T  I
)      CCC P     D  CCCIO T    _  F
)      KKO A        OOOAN I    P  T
)       2N D        NNNL  N    A  O
)          S                   D  U
)          [           _       S  T
)          1           I       [
)          5           N       4
)          :                   :
)          0                   0
)          ]                   ]
)      ...  ....  ........  -     -- .
)      bbb xxxx bbbbbbbb b    xx b
adq
)      0: 001 0000 10011100 0 > .. .
)      5: 011 0000 10011100 0 > .. .
)     10: 111 0000 10011100 0 > 00 .
)     15: 101 0000 10011100 0 > 00 .
)     20: 001 0000 10111100 0 > 00 .
)     25: 011 0000 10111100 0 > 00 .
)     30: 111 0000 10111100 0 > 00 .
)     35: 101 0000 10111100 0 > 00 .
)     40: 000 0000 10001000 0 > 00 .
)     45: 010 0000 10001000 0 > 00 0
)     50: 110 0000 10001000 0 > 00 0
)     55: 100 0000 10001000 0 > 00 0
)     60: 000 ffff 10011000 0 > 00 0
)     65: 010 ffff 10011000 0 > 00 0
)     70: 110 ffff 10011000 0 > 10 0
)     75: 100 ffff 10011000 0 > 10 0
)     80: 000 ffff 10001100 0 > 10 0
)     85: 010 ffff 10001100 0 > 10 0
)     90: 110 ffff 10001100 0 > 00 0
)     95: 100 ffff 10001100 0 > 00 0
)    100: 001 0000 10001000 0 > 00 0
)    105: 011 0000 10001000 0 > 00 0
)    110: 111 0000 10001000 0 > 00 0
)    115: 101 0000 10001000 0 > 00 0
)    120: 001 0001 10001000 0 > 00 0
)    125: 011 0001 10001000 0 > 00 0
)    130: 111 0001 10001000 0 > 01 0
)    135: 101 0001 10001000 0 > 01 0
)    140: 001 0012 10001000 0 > 01 0
)    145: 011 0012 10001000 0 > 01 0
)    150: 111 0012 10001000 0 > 02 0
)    155: 101 0012 10001000 0 > 02 0
)    160: 001 0122 10001000 0 > 02 0
)    165: 011 0122 10001000 0 > 02 0
)    170: 111 0122 10001000 0 > 03 0
)    175: 101 0122 10001000 0 > 03 0
)    180: 001 0123 10001000 0 > 03 0
)    185: 011 0123 10001000 0 > 03 0
)    190: 111 0123 10001000 0 > 04 0
)    195: 101 0123 10001000 0 > 04 0
)    200: 001 1234 10001000 0 > 04 0
```

Figure 3.16   Message Screen Style
Simulation Results

```
********************************************************************************
Chip: ~genpooler/pooler/DFT_CHIP                        Functional Simulator
- - - - - - - - - - - - - - - - - - - - - - - - - - Genesil Version v7.1- - - - - - - - - - - - - - - - - - - - - - - - - - -

OTIMEPNT  DATCON REFCON SPCON  IN_PADS OUT_PADS TESTIN SHIFTOUT
O 80       0      0      0      FFFF    10       0      0
O 85       0      1      0      F0F0    10       0      0
O 90       0      1      0      F0F0    08       0      0
O 95       0      1      0      F0F0    08       0      0
O 100      0      1      0      F0F0    08       0      0
O 105      1      0      0      F00F    08       0      0
O 110      1      0      0      F00F    08       0      0
O 115      1      0      0      F00F    08       0      0
O 120     *1     *0     *0     *F00F   *08      *0     *0

OTIMEPNT  CLOCK CLOCK2 MSK_ON SERIAL_IN dfout rfout mfout
O 80       0     0      1      0         0000  0000  FFFF
O 85       0     1      0      0         0000  0000  FFFF
O 90       1     1      0      0         0000  F0F0  FFFF
O 95       1     0      0      0         0000  F0F0  FFFF
O 100      0     0      0      0         0000  F0F0  FFFF
O 105      0     1      0      0         0000  F0F0  FFFF
O 110      1     1      0      0         F00F  F0F0  FFFF
O 115      1     0      0      0         F00F  F0F0  FFFF
O 120     *0    *0     *0     *0        *F00F *F0F0 *FFFF


          25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 10 10 11 11 12
                                                       0  5  0  5  0
          +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
CLOCK        +-----+      +-----+      +-----+      +-----+      +-----+
          --+     +-----+      +-----+      +-----+      +-----+      +--
CLOCK2    -----+      +-----+      +-----+      +-----+      +-----+
             +-----+      +-----+      +-----+      +-----+      +-----+
             +-----+      +-----+      +-----+      +-----+      +-----+
phase_ta  -----+      +-----+      +-----+      +-----+      +-----+
             +-----+      +-----+      +-----+      +-----+      +-----+
phase_tb     +-----+      +-----+      +-----+      +-----+      +-----+
             -----+      +-----+      +-----+      +-----+      +-----+
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

 INSERT  MESSAGES  GRAPHICS  FORM          OVERLAY        RECORD    UTILITY
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
BACK        QUERY          HIER_LEVEL    ENVIRONMENT    NEWSCREENS
BIND        CYCLE          RUN_VECTORS   SCROLL         PICK_SCREEN
ASSERT      STEP           UNBIND        FIGHTS         FORMAT_SCREEN
            PROPAGATE      VERIFY_VALUE
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .


>SIMULATION>
```

Figure 3.17   Formated Screen Style Simulation Results

77

functions were written. Basic building block type check functions were written to accomplish limited simulation tasks. These check functions are also convenient for use during manual interactive simulation sessions. High level check functions were then written to automatically accomplish a complete simulation test by sequentially calling a set of the basic check gunctions.

The following basic building block check functions were written and used during simulation on the DFT_CHIP:

1. **MSP** - function to set the mask register values in a parallel manner.

2. **RSP** - function to set the reference register values in a parallel manner.

3. **DSP** - function to set the data register values in a parallel manner.

4. **TSS** - function to serially load 32 bits of data into the scan path.

5. **TS** - function to serially load one to 32 bits of data into the scan path.

6. **SS** - function to serially shift data into the data, reference, and mask registers and optionally at the same time load data into the scan path.

7. **force_in** - function to force values from the scan path latches into the data latches of the testability latches.

8. **swap_in** - function to swap the contents of the data and scan path latches in the testability latches.

9. **sample_in** - function to sample and replace the scan path latch values with the data latch values in the testability latches.

10. **LSB** - function used to assign the next bit of data which will be serially loaded via the serial input pin.

11. **tog** - function to set up toggle patterns for the clock signals.

12. **untog** - function to untoggle the clock signals.

The complete GENIE language code written for these check functions is contained in Appendix A.

Complete functional simulation testing of the DFT_CHIP was accomplished using the following high level check functions:

1. **initall** - function to initialize the values present at all input and output pins of the chip.

2. **test_parallel_in** - function to test the chip during parallel load operations.

3. **test_serial_in** - function to test the chip during serial load operations.

4. **test_force** - function to check force operations after serial loads of the scan path.

5. **test_scan_clock** - function to test the ability to operate the local scan path clock at a faster rate than the global system clock and to test the ability to conduct scan path operations during periods that the global system clock is not operating.

Each of these functions includes the traceobj command to produce MASM test vector files. The complete code written for these check functions is contained in Appendix B. These check functions were used to verify the functional operation of the chip using both GFL and GSL models. The results from these simulation operations indicated that the DFT_CHIP would function as desired once it was fabricated.

2. Timing Analysis

Genesil's Timing Analyzer feature provides an easy to use means of evaluating the timing characteristics and

relationships of a chip design. It uses algorithms, which do not need or make use of test vectors, to produce reports on the following timing characteristics [Ref. 18:p. 1-1]:

    Maximum operating speed
    Speed limiting paths within the design
    Constraints on duty cycles
    Input setup and hold times
    Output delays
    Signal delays, setup, and hold times for internal nodes
    Path delays between internal nodes.

Through an examination of these reports changes can be made to a design to help insure that it will operate without timing problems. Also, this information can be used to optimize chip speed performance for a design which is required to operate in an environment needing a specific minimum operating frequency.

For the DFT_CHIP design the main use of the Genesil Timing Analyzer was to verify that there were no timing relationship conflicts and to determine the anticipated maximum operating frequency for both clock signals on the chip. Since chip size not operating frequency became an overriding design criteria, the timing information was not used to redesign the chip layout to provide increased chip speed. However, it was used to gain information on the maximum anticipated clock frequencies. These clock frequencies were then specified for the clock signals during the netlisting process. Doing this insured that the compilation process would produce a fabrication layout able to handle the maximum clock frequencies calculated by the Timing Analyzer

for the specific signal paths and internal components of the DFT_CHIP.

Three specific Timing Analyzer reports were looked at for the DFT_CHIP. First, the **CLOCKS** command was issued to generate a **Clock Report**. This report provides details on the maximum operating frequency and duty cycle limitations for a particular clock signal. Included is information on the minimum high phase times for the internal nonoverlapping phase signals derived from the clock signal, the minimum nonsymmetric and symmetric clock signal times, and details concerning the worst case paths for each of the phases of the clock signal [Ref. 18:p. 5-4]. Second, the **SETUP_HOLD** command was used to generate a **Setup and Hold Mode Report** which indicates the setup and hold times needed for each input signal relative to the falling edge of the clock signal [Ref. 18:p. 6-1]. Finally, the **VIOLATIONS** command was issued to produce a **Violations Report** which indicates if any internal hold time violations exist for the design configuration produced [Ref. 18:p. 11-1]. It is a Genesil <u>requirement</u> that a Violations Report be generated and checked for all Genesil chip designs prior to design tapeout for fabrication to insure that the design will not experience internal timing problems after return from fabrication [Ref. 18:p. 11-1].

Since there are two different clock signals utilized for the DFT_CHIP, all three timing analysis reports had to be run twice. Based on the information obtained from these

81

reports the maximum expected operating frequencies for the DFT_CHIP were 4.43 MHz for the global system clock signal CLOCK and 28.6 MHz for the local scan path clock signal CLOCK2. Appendix C provides the complete timing analysis reports generated for both clock signals.

D. AUTOMATIC TEST GENERATION AND FAULT COVERAGE

The usage of Genesil's **Automatic Test Generation** (ATG) feature was extremely important to the work done for this thesis. It allows an exploration into the need for and effects of including DFT features in a chip design. The ATG feature provides a method to automatically generate test vectors which will uncover defects that occur due to the manufacturing (not design) process. ATG looks at manufacturing defects in terms of stuck-at 0 and stuck-at 1 faults. By running ATG for an object under design, test vectors are developed to uncover these stuck-at faults. Through a process of fault grading, ATG also determines the fault coverage for the test vectors it develops. When properly enabled, the ATG feature will produce a set of maximum possible fault coverage test vectors for the current design configuration of the object being developed. Therefore, by examining the achievable fault coverage level for an object, the effects of including DFT features into a design are quickly determined. [Ref. 19:p. 1-1]

Genesil produces its fault coverage results through use of the classical D algorithm. This algorithm provides an algebra

for simplifying the computational tasks of testing for faults within a design. The D algorithm utilized by Genesil uses a process called **justification** to determine the inputs to apply to the design and a process called **sensitization** to check the outputs which will result for a specific sequence of events. [Ref. 19:pp. 1-2, 1-3]

Justification is related to the controllability of being able to set the inputs of a gate to specific values. The ATG justification process places desired values on the gate under test and then tries to back these values out of the circuit to primary inputs. If this process of backing the values out of the design is successful without producing any conflicting values on the primary inputs then the stuck-at test for that gate is said to be justified and the inputs to the gate are determined to be controllable.[Ref. 19:p. 1-4]

Sensitization is closely related to the observability of a circuit. It is the process of propagating the output of a gate being tested to a primary output of the circuit. If this is accomplished then the stuck-at test for the gate is said to be sensitized and the design configuration allows for observability of the output node of the gate.[Ref. 19:p. 1-4]

During the ATG process, Genesil first accomplishes an initial pass over the design to locate any faults which are obviously untestable and also to develop a testing priority to conduct tests first on those nodes which have the highest degree of observability and controllability. To speed the

test process, ATG uses a modified breadth-first search to look simultaneously at multiple paths to the primary inputs and outputs. ATG takes a single step along a given path during an evaluation of justification or sensitization, checks and puts the results for that path and step onto a list of pending processes, and then moves to the next path possible for that particular step. This causes a gradual expansion of the test process at each step. By doing this rather than checking just one path at a time, conflicts caused by any incompatible assertions, which may result during the justification and sensitization process, are recognized quicker. This helps to speed up the rate at which the overall ATG process can be accomplished.[Ref. 19:p. 1-7]

The penalty paid for the speed up possible from doing the breadth-first search is that if a conflict is found ATG has to retrace its steps all the way to the test assertion that caused the conflict. Then, all justification and sensitization evaluations must be repeated between the point of the assertion and the location at which the conflict was discovered. In doing this, paths not related to the conflict also are forced into being rechecked. To minimize the amount of work that needs to be redone, ATG partitions assertions into different search groups that it organizes based on a determination of which portions of the circuit are independent from each other. Therefore, a conflict in one search group does

not extend the need to rework justifications or sensitizations into another search group.[Ref. 19:p. 1-8]

Since the test algorithm used by the ATG process is trying to back justifications out of or propagate sensitizations through a circuit, sequential logic may present problems during the development of a test strategy. To handle sequential circuits the ATG process utilizes a method called **time unrolling**. This method has the effect of translating sequential circuit elements into combinational circuit elements that are examined over a restricted time range. This translation process can be viewed as producing a three dimensional set of combinational circuits where the third dimension is related to the number of timepoints that originate from the feedback path of the sequential circuit. The time window for the number of timepoints considered during the ATG process may be either a default based on Genesil's determination of the sequential depth of individual nodes in the object or it may be limited to a user specified number. The higher the number the slower the ATG process will proceed, but a higher number may produce a greater fault coverage or a denser maximum fault coverage test vector set.[Ref. 19:pp. 1-6, 1-7]

To run the ATG process on an object the following parameters from the ATG Control section of the ATG form must first be specified:

1. **Output File** - specifies the file to which the test vectors produced by ATG will be written.

2. **Sequential Depth** - determines the maximum number of timepoints which can be used to try and instantiate a specific fault test. Using the parameter of -1 provides a default for each node based on the shortest path from the node to a primary output.

3. **Random Input Vectors** - determines the number of random seed vectors the ATG process will generate and use as inputs. Using random input vectors may speed up the ATG process. Choosing a parameter of -1 will provide the same number of random vectors as the maximum sequential depth found in the object.

4. **Initialization Vectors** - determines the number of vectors from the Input File which will be run as simulation only to provide an initialization effect for the object. These vectors will <u>not</u> be included in the output test vector file. Any remaining vectors from the Input File will have the normal ATG process run on them.

5. **Default Toggles** - determines if the default toggle definition will be used for the clock signal in the circuit. Selecting NO will cause the clock toggle definition to be obtained from the Startup File if it exists.

6. **Limit Time** - determines the time limit (in CPU time usage) for which the ATG process will run. Selecting NO specifies unlimited time (although the ATG process may still quit if it is not able to instantiate a fault test after running a large number of test vectors).

7. **Limit Coverage** - selecting YES and specifying a fault coverage value will halt the ATG process when that fault coverage value has been obtained. Selecting NO indicates a default of 100 percent fault coverage.

8. **Fault Grade Only** - selecting YES allows the previously generated set of MASM test vectors specified as the Input File to be fault graded.

9. **Enable Input File** - selecting YES and specifying a file name provides the file to be used during either the initialization process or the file to be fault graded.

10. **Enable Startup File** - <u>must</u> be selected as YES and have the Startup File name provided for cases where the Default Toggles parameter is set to NO. The Startup File contains the clock toggle definitions which will be used to replace the default toggle definition. <u>All</u> chips with two or more clocking regimes <u>must</u> specify the clock toggle definitions in a Startup File. Reference

86

19, Appendix B provides examples of the toggle defini-
tions which can be written for chips with multiple
clocking regimes.

11. **Enable DFT File** - determines if a DFT file will be used
to specify artificial primary inputs, outputs, and no
connects. This feature allows the designer to specify
additional inputs and outputs at internal circuit
locations during the initial design process to help
determine if including DFT features will raise the fault
coverage possible. Select <u>NO</u> for this parameter for
final chip testing using the ATG process.

12. **Enable Coverage In** - selecting YES causes the ATG
process to consider a coverage map from a previous ATG
run when it starts the present run. Coverage maps are
files which provide the ATG process information on which
faults remain to be tested in a design.

13. **Coverage Output File** - specifies the name of the
coverage map file to which coverage information will be
written.

Once all ATG Control section parameters are specified the ATG

process is started by selecting **RUN_ATG** from the menu of the

ATG form.[Ref. 19:pp. 4-4 - 4-7]

Once the ATG process is running its current status can be

displayed in the ATG Status section of the ATG form. To

update the status to the present time select the **UPDATE_SCREEN**

choice from the ATG form menu. As an alternative, the status

can be continuously updated and displayed by issuing the

command **update_loop** at the ATG form command line prompt

[Ref. 19:p. 4-11]. The ATG process runs in the background.

This allows other Genesil activities to be performed while

waiting for ATG to complete unless the status is continually

being updated via the update_loop command [Ref. 19:p. 4-2].

Figure 3.18 shows a complete screen view of the ATG form, to

```
****************************************************************************
Chip: ~genpooler/pooler/DFT_CHIP                        ATG Control Program
-------------------------------Genesil Version v7.1-------------------------------
                              ATG Control

      Output File:              > maxcov_example_____
      Sequential Depth:         > -1_____
      Random Input Vectors:     > -1_____
      Initialization Vectors:   > 0_____
      Default Toggles:          [NO] YES
      Limit Time:               [NO] YES
      Limit Coverage:           [NO] YES
      Fault Grade Only:         [NO] YES
      Enable Input File:        [NO] YES
      Enable Startup File:       NO [YES]
      Startup File:             > clocks_____
      Enable DFT File:          [NO] YES
      Enable Coverage In:       [NO] YES
      Default Coverage Out:      NO [YES]


                              ATG Status

      Vector              Tests              CPU Time (h:m:s)
      ------     ------------------------    ------------------

                 Change   Tested   Percent   Change    Total

      10         9        224      10.48     10        1:37
      11         5        229      10.71     23        2:00


                            Command Status

      ATG running

----------------------------------------------------------------------------
 INSERT  MESSAGES  GRAPHICS  [FORM]   OVERLAY           RECORD      UTILITY
----------------------------------------------------------------------------
NOP                CHECK_FORM      RUN_ATG          DUMP_COVERAGE   UPDATE_SCREEN
ACCEPT_FORM        SAVE            HALT_ATG         EDIT_STARTUP    VIEW_LOG
PIGEONHOLE         TEXT_SPEC       KILL_ATG         EDIT_DFT        ANALYZE
CANCEL                             ENABLE_CURRENCY
----------------------------------------------------------------------------

>ATG>
```

Figure 3.18   ATG Form Screen Display

88

include the ATG Control and Status sections, for an example case of running ATG on the DFT_CHIP.

Upon completion of the ATG process (through either expiration of the specified time limit, achievement of the specified fault coverage level, or completion of fault grading for an input test vector set) the final results can be examined in depth by selecting the **ANALYZE** command from the ATG form menu. If the ATG process has not yet finished, it can be forced to stop by choosing the **HALT_ATG** command from the ATG menu. The ANALYZE command will then provide results based on all tests made prior to the halt being issued. Once ANALYZE is chosen a new menu is presented which allows the fault coverage statistics to be examined from the top level of chip down to the individual gate level. At the gate level the individual stuck-at faults which have or have not been tested are listed. Examination of this information provides the details on what portions of the design contains gates which the ATG process has not been able to fault test. Based on this information the design can be modified or DFT features can be included to raise the fault coverage obtainable. Figure 3.19 is an example for the DFT_CHIP of the format and type of information which can be provided by using the ANALYZE command.[Ref. 19:p. 5-2]

The starting use of the ATG feature was to look at the characteristics of the BASIC_CHIP design. First, the ATG process was run, with no limits on either time or fault

```
FAULTS
) / (module):  2003 tests out of 2137 (93.7295%)
)    combiner2 (module):  74 tests out of 80 (92.5%)
)       AND5 (module):  2 tests out of 3 (66.6667%)
)          and (AND):  2 tests out of 3 (66.6667%)
)       XOR6 (module):  3 tests out of 4 (75%)
)          xor (XOR):  3 tests out of 4 (75%)
)       XOR4 (module):  2 tests out of 4 (50%)
)          xor (XOR):  2 tests out of 4 (50%)
)       XOR2 (module):  3 tests out of 4 (75%)
)          xor (XOR):  3 tests out of 4 (75%)
)       XOR0 (module):  3 tests out of 4 (75%)
)          xor (XOR):  3 tests out of 4 (75%)
)    serial_input (module):  2 tests out of 4 (50%)
)       idatala_b[0] (LATCHM):  2 tests out of 4 (50%)
)    ref_in (module):  240 tests out of 259 (92.6641%)
)       refcontrol (module):  2 tests out of 3 (66.6667%)
)          AND0 (module):  2 tests out of 3 (66.6667%)
)             and (AND):  2 tests out of 3 (66.6667%)
)       ref1 (module):  118 tests out of 128 (92.1875%)
)          DFF6 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF1 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF5 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF0 (module):  9 tests out of 12 (75%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)             data_y[0] (LATCHM):  2 tests out of 4 (50%)
)          DFF4 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF2 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF3 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF7 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)       ref2 (module):  120 tests out of 128 (93.75%)
)          DFF10 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF9 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF13 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF8 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF12 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF14 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF11 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)          DFF15 (module):  11 tests out of 12 (91.6667%)
)             ctl_y[0] (LATCHM):  3 tests out of 4 (75%)
)    load (module):  2 tests out of 4 (50%)
)       idatala_b[0] (LATCHM):  2 tests out of 4 (50%)
)    tlatch32 (module):  683 tests out of 706 (96.7422%)
)       TLATCHL1 (module):  342 tests out of 353 (96.8839%)
)          a (NOR):  2 tests out of 3 (66.6667%)
)          ca (LATCHM):  2 tests out of 4 (50%)
)          cb (LATCHM):  2 tests out of 4 (50%)
)          cf (LATCHM):  2 tests out of 4 (50%)
)          cs (LATCHM):  2 tests out of 4 (50%)
)          ld_y[0] (LATCHM):  2 tests out of 4 (50%)
)       TLATCHL0 (module):  341 tests out of 353 (96.6006%)
)          a (NOR):  2 tests out of 3 (66.6667%)
)          ca (LATCHM):  2 tests out of 4 (50%)
)          cb (LATCHM):  2 tests out of 4 (50%)
)          cf (LATCHM):  2 tests out of 4 (50%)
```

Figure 3.19   Fault Coverage Information Obtainable
from using ANALYZE Command

90

coverage, to determine the maximum fault coverage obtainable for the design without DFT features. Once the test run reached a stage where additional test vectors were producing no increase in fault coverage the program was halted and the coverage obtained was examined using the ANALYZE command. Table IV presents the fault coverage results for BASIC_CHIP design in terms of its individual modules. Also included is the number of test vectors needed to achieve the fault coverage listed. The results listed for $dummy come from a dummy module that the ATG process creates to contain artificial/dummy constructs used during the evaluation of certain types of Genesil blocks [Ref. 19:p. E-5]. These dummy constructs are needed to account for differences between the models generated by the Genesil simulator and the additional internal nodes created by the mapping of these models into the primitives used by Genesil during the ATG process. Therefore, the $dummy results need to be included and considered when evaluating the fault coverage of a complete, top-level design.

As can be seen from Table IV, the modules with the lowest fault coverage were the combiner, input pad and $dummy modules. Since the $dummy module is based on a mapping of the simulation models for Genesil blocks to the primitives used by ATG, there is no way to identify how to improve this module's fault coverage without changing the components of the circuit design. Similarly, the fault coverage for the input pad modules is a function of the pad specifications and was not

TABLE IV
ATG FAULT COVERAGE RESULTS FOR BASIC_CHIP DESIGN

Module	Faults Tested	Fault Coverage %
adder	146 of 155	94.1935
combiner	160 of 192	83.3333
data_in	240 of 259	92.6641
mask_in	240 of 259	92.6641
output	15 of 15	100.0000
ref_in	240 of 259	92.6641
xnorreg	112 of 112	100.0000
input pads (all)	46 of 88	52.2727
output pads (all)	20 of 20	100.0000
$dummy	8 of 44	18.1818
BASIC_CHIP Total	1227 OF 1403	87.4555
Total test vectors used: 530		

observed to change unless the type of input pads utilized was changed.

For the combiner module the ANALYZE command was selected and this module was looked at down to the gate level to try and determine which gates were not being completely tested. Based on this examination it was found that each of the four identical combiner module sections, used to convert a 4-bit wide slice of results coming form the xnorreg module output to a 3-bit number representing the sum of correlated bits, had the same gates with faults not tested. These are the gates labeled OR0, OR1, XOR4, XOR5, AND4 and AND5 in Figure 3.4.

Knowing which gates were not being completely tested, a decision could be intelligently made about where to place a scan path to help increase fault coverage for the combiner module. As previously discussed, the DFT_CHIP design was made by inserting a scan path made from testablilty latches in between portions of the original combiner module. As shown in Figure 3.5, the insertion location of the scan path in the combiner module was just prior to the incompletely tested gates along the direction of circuit propagation. It was hoped that by being able to scan in specific test vector sequences the ATG process would now be able to fault test some of the previously untested gates.

The ATG feature was then used to determine the fault coverage possible for the DFT_CHIP design. The results for the ATG test run, obtained using the ANALYZE command as per

Figure 3.19, are shown in Table V. As indicated, the inclusion of the scan path had a positive effect in several areas. First, the fault coverage for those gates not completely tested in the combiner module of the BASIC_CHIP design went up. The DFT_CHIP design was now able to test for faults oo all of the gates from the combiner1 module and 92.5 percent of the gates from the combiner2 module. Since the sum of these two modules contains the same exact gates found originally in the combiner module of the BASIC_CHIP design, the result of including the scan path was to increase the fault coverage for the total combiner gates from 83.3333 per ent to 96.8750 percent. Next, the scan path was able to raise the fault coverage for both the adder module (which is downstream in terms of circuit propagation from the scan path) and the $dummy module to 100 percent. Most importantly, the overall fault coverage of the complete chip was raised from 87.4555 percent in the BASIC_CHIP design to 93.7295 percent in the DFT_CHIP design. Finally, the inclusion of the scan path not only raised the fault coverages obtainable but also lowered the number of test vectors needed to obtain the maximum fault coverage. It took 530 test vectors to obtain the maximum possible fault coverage for the BASIC_CHIP design but only 363 test vectors for the DFT_CHIP design.

Based on these results, the inclusion of the scan path had a considerable effect on the testability of the correlator chip. Due to the decreased number of test vectors needed to

TABLE V
ATG FAULT COVERAGE RESULTS FOR DFT_CHIP DESIGN

Module	Faults Tested	Fault Coverage %
adder	155 of 155	100.0000
combiner1	112 of 112	100.0000
combiner2	74 of 80	92.5000
combiners 1 and 2	186 of 192	96.8750
data_in	240 of 259	92.6641
mask_in	240 of 259	92.6641
output	15 of 15	100.0000
ref_in	240 of 259	92.6641
tlatch32	683 of 706	96.7422
xnorreg	112 of 112	100.0000
input pads (all)	56 of 104	53.8462
output pads (all)	24 of 24	100.0000
$dummy	52 of 52	100.0000
DFT_CHIP Total	2003 of 2137	93.7295
Total test vectors used: 363		

obtain maximum possible fault coverage, the time used for testing of the DFT_CHIP after fabrication should be minimized. Perhaps more importantly, the increase in the overall fault coverage should lower the defect level found among chips that successfully pass the testing process. Using equation (1.7) as a basis for determining defect level, Figure 3.20 shows a plot of the defect levels for varying yields using the maximum fault coverage percentages of the BASIC_CHIP and DFT_CHIP designs. This shows graphically the gain, in terms of lower defect levels for the same yield, obtained from including a scan path in the design.

It should be noted that there are also two main penalties incurred by including the scan path which are representable of the effects of including a DFT structure into any design. First, the scan path itself introduces gates into the design which then have to be fault tested. The difference between the BASIC_CHIP design and the DFT_CHIP design was the inclusion of gates resulting in an additional 734 faults to test. For the correlator chip design this was over a 50 percent increase. A more complex design would have a smaller percentage increase for a scan path with the same number of testability latches but it could still be significant. A more complex design might also need more testability latches to produce a scan path able to raise fault coverage to a desired level. Secondly, the inclusion of a DFT feature like a scan path incurs a size penalty for the chip layout due to the
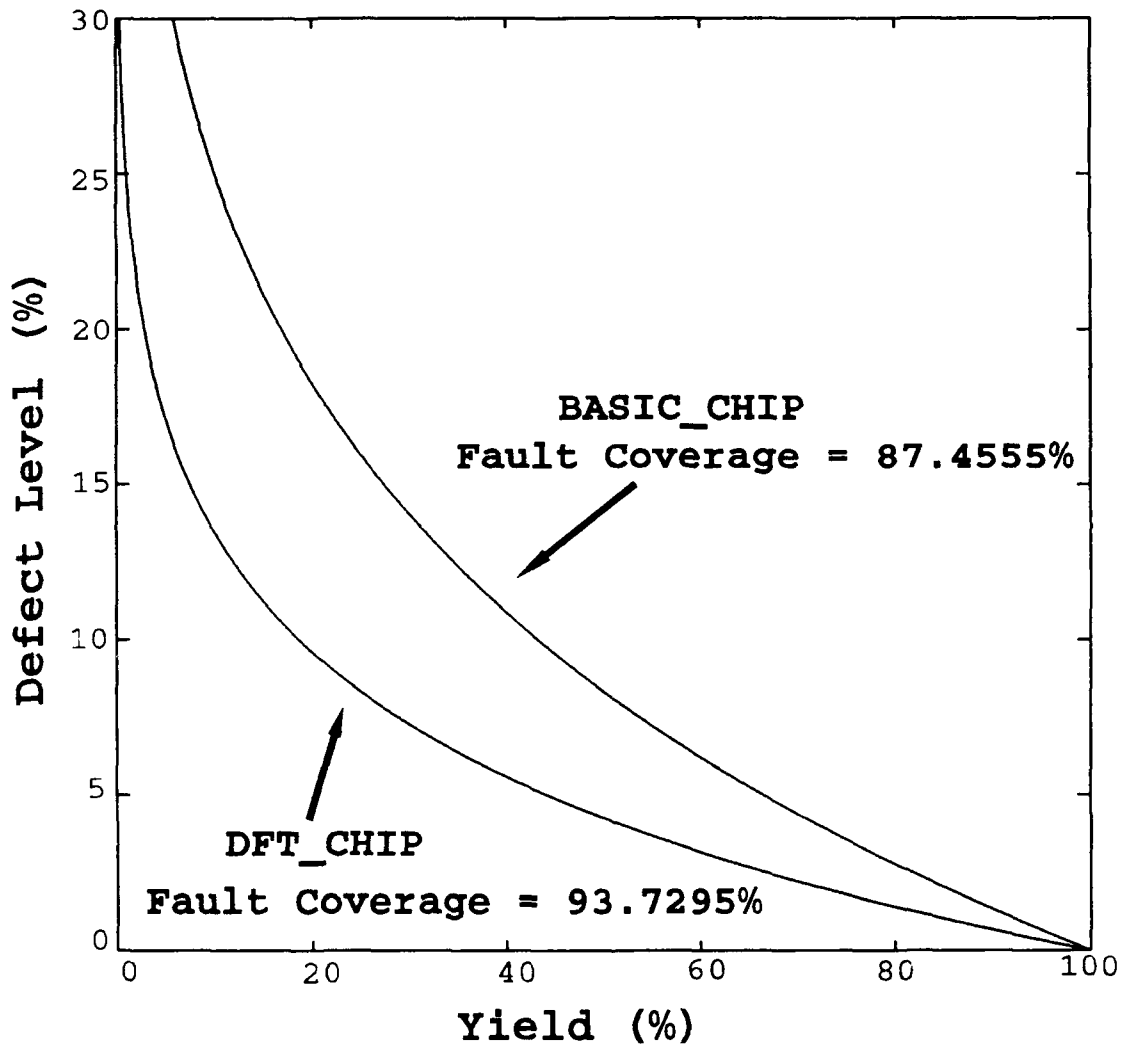
Figure 3.20 Defect Level versus Yield Using Maximum
Fault Coverages of the DFT_CHIP and BASIC_CHIP Designs

additional gates needed. Since the BASIC_CHIP design was not optimized for size the same way the DFT_CHIP design was, no direct comparison can be made for these two cases. However, the size increase will be related to the number of additional gates used to include a DFT feature. Therefore, within an order of magnitude, the percentage increase in gates should be related to the percentage increase in total chip size. Additional penalties which might be experienced for including a DFT feature like a scan path include additional power consumption and possible lower limits on maximum clock speed.

The final use of the ATG feature was to look at the fault coverage levels obtainable from the test vectors used to check the functionality of the DFT_CHIP desigo. Each of the MASM test vector files produced during the simulation process using the four high level check fuoctions (test_parallel_in, test_serial_in, test_force, and test_scan_clock) were fault graded using the ATG feature. Table VI shows these results. The results show that test vectors which are used to check the proper logic or functionality operation of a chip may not be very good for insuring a high degree of fault coverage. This demonstrates the need to test a chip using test vector sets which both look at the functionality issue and those which provide the maximum possible degree of fault coverage.

TABLE VI
ATG FAULT COVERAGE RESULTS FOR DFT_CHIP USING
CHECK FUNCTION PRODUCED TEST VECTOR SETS

Check Functions	Faults Tested	Fault coverage %
test_parallel_in	848 of 2137	39.6818
test_serial_in	929 of 2137	43.4722
test_force	725 of 2137	33.9261
test_scan_clock	408 of 2137	19.0922

IV. FABRICATION AND TESTING

Once a Genesil chip design has been completed (including all aspects of floorplanning, simulation testing, timing analysis, ATG processing and final compilation) it is ready to be sent for fabrication. This chapter first examines the steps taken to have the DFT_CHIP design fabricated through the MOSIS Service. Upon completion of fabrication, chips need to be tested to both validate that no functional errors exist in the design and that no manufacturing errors occurred during the fabrication. The second section of this chapter provides details on the test gear and testing process used to conduct this testing on fabricated copies of the DFT_CHIP design.

A. FABRICATION METHODOLOGY

The DFT_CHIP design produced for this thesis was fabricated through use of the MOSIS Service. MOSIS stands for MOS Implementation System and the MOSIS Service provides fabrication services to university classes, government agencies and government contractors under sponsorship of the Defense Advanced Research Projects Agency (DARPA) and the National Science Foundation (NSF) [Ref. 20:p. 3]. The MOSIS Service provides an inexpensive means of fabrication for standard cell and full-custom VLSI designs using 3.0, 2.0, 1.6 and 1.2 micron double metal CMOS technologies [Ref. 20:p. 1]. Multiple fabrication line vendors are utilized to manufacture

designs, and costs are kept down for MOSIS users by combining projects from several users onto a single wafer during fabrication runs [Ref. 20:p. 1]. Instead of paying between $50,000 and $80,000 for a complete wafer lot to be produced, users pay for only that percentage of the silicon space on the wafer that their designs occupy resulting in chips which can be fabricated for as little as $400 [Ref. 20:p 1].

Actual charges for chip fabrication depend on which of the four MOSIS size categories the chip falls into. The four MOSIS size categories and the maximum sizes allowed for them are: tiny (2.3mm by 3.4mm), small (4.6mm by 6.8mm), medium (6.9mm by 6.8mm) and large (7.9mm by 9.2mm). As previously discussed, the DFT_CHIP design was limited to fitting within the MOSIS small size category due to budget constraints.

For Genesil designed chips being sent to MOSIS for fabrication there are two steps which must be taken prior to final compilation that differ from normal Genesil design practices. The first step involves the placement of pads for the chip during the pinout process of floorplanning. MOSIS highly recommends that designs should have the same number of pads located on all four sides of the cavity well, and that along each side the pads should be spaced approximately the same distance apart. This placement configuration may be slightly stricter than that encountered for designs not being fabricated through MOSIS. MOSIS requires this type of placement to insure that the bonding wires to the pads will

.ot be excessively long, cross, or be at too great a bonding angle. Secondly, a Genesil chip design sent to MOSIS should have the **NO_PACKAGE** option chosen for it in Genesil vice choosing a package type and attaching the bonding wires during pinout as is normally done.

MOSIS will have all chips submitted for fabrication packaged and will provide a bonding diagram with the fabricated chips. The package type used by MOSIS depends on the number of pins the submitted chip design has. Dual In-line Packages (DIP) are used for 28, 40, or 64 pin designs, and Pin Grid Array (PGA) packages are used for 84, 108, or 132 pin designs [Ref. 20:p. 57]. Additional information on MOSIS pad placement requirements and packaging practices is contained in Chapter 9 of Reference 20.

Once a Genesil design has been completed there are several additional tasks which need to be accomplished before submitting the design to MOSIS f r fabrication. First, an account must be established with MOSIS to pay for the fabrication services. Universities which teach VLSI design may apply for government sponsored funding of their VLSI design projects [Ref. 20:p. 12]. This funding method was used for the DFT_CHIP design with funding being obtained via the NSF.

Secondly, the MOSIS fabrication schedule must be checked to find a fabrication run being done using a fabrication line technology available in Genesil. MOSIS fabrication runs occur approximately every two weeks with runs alternating between

3.0 and 2.0 micron feature sizes and p-well and n-well technologies. MOSIS has 1.6 and 1.2 micron feature size runs on-demand as enough projects to fill the runs are received.

Genesil offers over 20 different combinations of fabrication line vendors, feature sizes (ranging from 1.0 to 3.0 micron), and n-well or p-well process to choose from. A match must be made between a vendor, feature size and process type of a fabrication run scheduled by MOSIS and the same combination from among the Genesil choices. Since the MOSIS fabrication schedule does not list the fabrication line vendor which will produce a run, a phone call must be made to MOSIS to obtain this information.

Having determined a match between a technology combination available in Genesil and also scheduled by MOSIS, the Genesil design must be completely recompiled in the chosen fabrication line technology if a different technology was previously specified. It should be noted that this can have an effect on the size and operating speed of a design so ideally a match should be determined early in the design process. Doing so will avoid the need to review the chip parameters once a chip design is completed. If a change must be made for the fabrication line technology, the only penalty within Genesil is the several hours of CPU time needed to completely recompile the chip. No other changes are needed within Genesil to change from one fabrication line technology to another. The compiled design layout produced by Genesil will utilize the

specific design rules for the fabrication vendor of the fabrication line technology chosen.

Upon completion of these steps the Genesil design must be exported from the Genesil system into a file which can be sent to MOSIS. To export a Genesil design the following steps need to be taken from within the Genesil operating environment.

1. Select the chip to be exported as the current object and return to the Genesil main menu.

2. Select **TOOLING** and then **TAPEOUT** from the Genesil menus.

3. Select **SIZING** or **NO_SIZING**. SIZING produces foundry customized data based on the design rules of the fabrication line foundry specified while NO_SIZING produces a generic type layout [Ref. 15:p. 11.18]. Not all fabrication line vendors utilize sized layouts. Since the present documentation does not provide information on which fabrication line technologies need to be sized, a phone call to the Genesil Silicon Compiler Corporation may be needed.

4. Choose **CIF** (Caltech Interchange Format) as the file format to be used to specify the layout. Genesil also offers GDSII (Calma Corporation GDSII Stream Format) but MOSIS requires submissions be done using the CIF format.

5. Specify the filename for the CIF file. Genesil will then create the CIF file within the Genesil environment as either a type HCF file (for unsized CIF) or a type CIF file (for sized CIF).

6. Export the file from the Genesil environment by selecting **ANCILLIARY_FILE**, **EXPORT_FILE**, choosing the file to export, and providing a filename to which the exported file will be written.

These steps will result in a CIF file of the chip design being exported to the UNIX directory from which Genesil was started.

The final step to be taken before submitting the design is to run the CIF file through the MOSIS **CIF_CHECKSUM** program which can be obtained from MOSIS in "generic" C source code

format. The purpose of the CIF_CHECKSUM program is to compute a checksum for the CIF file to help insure the accuracy of transmission process used to send the file to MOSIS. Due to the large size of CIF files (918K for the DFT_CHIP design) and the catastrophic result of errors, MOSIS requires that the checksum value be computed and provided along with all design submissions.

All interaction with MOSIS is normally done using electronic mail via the INTERNET computer network. Electronic mail correspondence to MOSIS must be done using formatted messages. This includes requests for information, identification of project information, submission of CIF files and requests for project status. Chapters 4 and 13 of Reference 20 give specifics on the means and message formats which must be used during communications with MOSIS.

Genesil produced chips have layouts based on specific fabrication line design rules vice the MOSIS scalable design rules. The project information message must indicate that specific fabrication line design rules were used by stating the fabrication line technology chosen (VTI_SCI for the DFT_CHIP which used a VLSI Technologies, Inc. fabrication line) on the **TECHNOLOGY** line of the message. This information should also be noted by including a statement in the **ATTENTION** field that the chip was produced using fabrication line specific design rules.

To fabricate the DFT_CHIP design through MOSIS, all the steps discussed in this subsection were followed. Based on the MOSIS fabrication schedule, the VTI-CN20A 2.0um n-well process from VLSI Technology, Inc. was chosen as the fabrication line technology for the chip. Based on this choice, the key parameters listing developed by Genesil for the DFT_CHIP design is shown in Figure 4.1. MOSIS was able to fabricate and return the DFT_CHIP design within eight weeks of the original design submission. The cost of chip fabrication was $2200 for 12 copies of the chip. Figure 4.2 is a reproduction of a picture taken by MOSIS of the actual chip which was fabricated.

B. TESTING METHODOLOGY

The final step undertaken in the development process for this thesis was the physical testing of the chips which were fabricated via MOSIS. Only by actually applying signals to a chip and observing its outputs can the functionality requirements and fabricated implementation of the design be finally validated. During the testing process, test vector sets were applied to the fabricated DFT_CHIP design to both verify the operational functionality of the chips and to check for improper chip operation which could have originated from errors during the fabrication process. Testing for operational functionality was used to validate the Genesil simulation results previously obtained for the chip. Testing for any manufacturing errors which occurred during fabrication was

106

```
) Key Parameters for Chip ~genpooler/pooler/DFT_CHIP
) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
)
) TIME ■ Thu Jun  7 17:39:59 1990
)
) ROUTE_VERSION ■ 87.20
) HEIGHT ■ 260.9 MILS
)     ( ■ 6626.85 u )
) WIDTH ■ 179.1 MILS
)     ( ■ 4549.14 u )
) ROUTED ■ 1 (0■NO,1■YES)
) TOTAL_WIRE_LENGTH ■ 83352 MILS
)     ( ■ 2117140. u )
) CORE_AREA ■ 24749.0 SQUARE_MILS
)     ( ■ 15967065. u2 )
) PADRING_AREA ■ 21981.0 SQUARE_MILS
)     ( ■ 14181262. u2 )
) PAD_AREA ■ 17934.7 SQUARE_MILS
)     ( ■ 11570750. u2 )
) ROUTE_AREA ■ 12936.0 SQUARE_MILS
)     ( ■ 8345789.9 u2 )
) PERCENT_ROUTING_OF_CORE ■ 52 %
) PERCENT_ROUTING_OF_CHIP ■ 27 %
) PERCENT_CORE_OF_CHIP ■ 52 %
) PERCENT_PADRING_OF_CHIP ■ 47 %
) PERCENT_PAD_OF_PADRING = 81 %
)
) NETLIST_VERSION ■ 1.0
) NETLIST_EXISTS ■ 1 (0■NO,1■YES)
)
) PHASE_A_TIME ■ 113.0 NANOSECONDS
) PHASE_B_TIME ■ 40.0 NANOSECONDS
) SYMMETRIC_TIME ■ 225.9 NANOSECONDS
) NUMBER_OF_TRANSISTORS ■ 5045
) POWER_DISSIPATION ■ 79.04 MILLIWATTS_@5V_10MHZ
)
) ROUTE_ESTIMATE_LVL ■ 0
) FLAT_ROUTE ■ 0 (0■NO,1■YES)
) TECHNOLOGY_NAME ■ CMOS-1
) PACKAGE_SPECIFIED ■ 1 (0■NO,1■YES)
) PACKAGE_NAME = NO_PACKAGE
) FABLINE_NAME ■ VTI_CN20A
) COMPILER_TYPE = GCX
)
) FLOORPLAN_VERSION ■ 7.1
) BOND_PAD_CNT ■ 40
) HEIGHT_ESTIMATE ■ 216.89 MILS
)     ( ■ 5509.006 u )
) WIDTH_ESTIMATE ■ 174.04 MILS
)     ( ■ 4420.615 u )
) FUSED ■ 1 (0■NO,1■YES)
) FUSION_REQUIRED = 1 (0=NO,1=YES)
) PINOUT ■ 1 (0■NO,1■YES)
) PINOUT_REQUIRED ■ 1 (0■NO,1■YES)
) PLACED ■ 1 (0■NO,1■YES)
) PLACEMENT_REQUIRED ■ 1 (0■NO,1■YES)
)
) DOWN_BONDS_ALLOWED ■ 1 (0■NO,1■YES)
) PKG_PIN_COUNT ■ 40
) OBJECT_TYPE ■ Chip
) AREA_PER_TRANSISTOR ■ 9.262081 SQUARE_MILS
)     ( ■ 5975.52438 u2 )
) PHYSICAL_IMPLEMENTATIONS_EXIST = 0 (0=NO,1=YES)
) CHECKPOINTS_EXIST ■ 0 (0=N^,1=YES)
) CAN_SET_FABLINE ■ 1 (0■NO,1■YES)
```

Figure 4.1   Keyparameters Listing for DFT_CHIP
Design Submitted for Fabrication
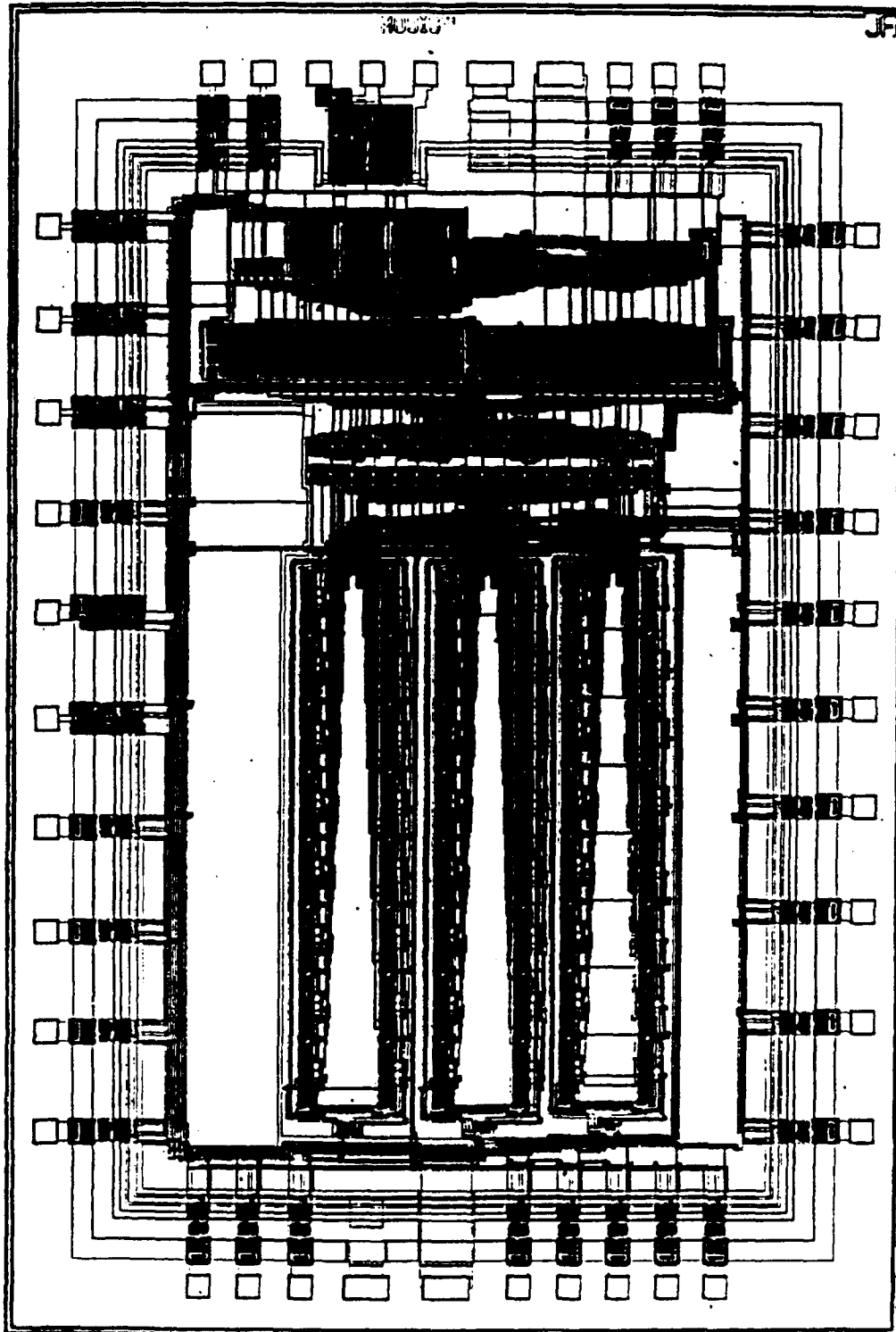
Figure 4.2   View of Fabricated DFT_CHIP

accomplished by applying the maximum fault coverage test vector set produced by the Genesil ATG process. Finally, testing was done on the chips to try and determine the maximum clock speeds which could be used for the fabricated DFT_CHIP design.

1. Testing Methodology for the DFT_CHIP Design

This subsection deals with the methodology used during the testing of the fabricated chips from the DFT_CHIP design. However, the approach taken and steps used is applicable to almost any other chip design produced using Genesil. The actual test facilities and test gear used to accomplish testing of the fabricated DFT_CHIP design chips involved usage of a commercial level chip tester, the **Textronix Digital Analysis System (DAS) 9100**, and its associated personal computer (PC) controller software.

The DAS 9100 is a commercial level tester which has numerous options for performing selected pattern generation and data acquisition functions and can be used in a stand-alone mode to accomplish chip testing. DAS 9100 Pattern Generation modules are used to apply input signal patterns to the device under test (DUT). DAS 9100 Data Acquisition modules are used to acquire the output results form the DUT and to monitor the input patterns supplied by the Pattern Generation modules.

Since testing on the DFT_CHIP was not done using the DAS 9100 in a stand-alone mode, its use in this manner will not be discussed here. Instead, the DAS 9100 documentation of

Reference 21 and Reference 22 should be reviewed for information concerning the configurations, capabilities, usage and setup of the DAS 9100 in a stand-alone mode. Additionally Chapter III of Reference 23 should be read to gain information on usage of the DAS 9100 configuration available at the Naval Postgraduate School (NPS).

Testing of the DFT_CHIP design was done exclusively by using the DAS 9100 in conjunction with its PC controller based **9100 Device Verification Software (91DVS)**. The advantage of performing testing in this method vice using the DAS 9100 in a stand-alone mode is the ease of use and speed of configuration setup possible for performing testing on the DAS 9100. Through use of a menu driven interface, the 91DVS software allows the user to send setup commands to the DAS 9100, apply test sequences to run tests, compare acquired test results against expected results and view the applied test patterns and acquired test results outputs along with the expected results on a screen display [Ref. 24:p. 1-1]. 91DVS can be run on any AT or XT configuration PC with DOS 3.0 or higher, and the PC is linked to the DAS 9100 via use of a GPIB interface card which controls a high-speed data link [Ref. 24:p. 1-1]. Figure 4.3 shows a functional block diagram of the NPS configuration for the test system utilizing the 91DVS software with the DAS 9100 tester.

The 91DVS software contains two software program choices for installation: DVS25 and DVS50. Based on the use

```
+--------------------------------------------------+
|         Personal Computer (AT/XT)                |
|   +------------------------------------------+   |
|   |           91DVS Software                 |   |
|   +------------------------------------------+   |
|                      ↕                           |
|   +------------------------------------------+   |
|   |         GPIB Interface Card              |   |
|   +------------------------------------------+   |
+--------------------------------------------------+
                       ↕
+--------------------------------------------------+
|              DAS 9100 Tester                     |
|   +----------------+   +----------------------+   |
|   |  91A32 Data    |   |  91S16 and 91S32     |   |
|   |  Acquisition   |   |  Pattern Generation  |   |
|   |  Module        |   |  Modules             |   |
|   +----------------+   +----------------------+   |
+--------------------------------------------------+
          ↑                        ↓
+--------------------------------------------------+
|              Device Under Test                   |
+--------------------------------------------------+
```
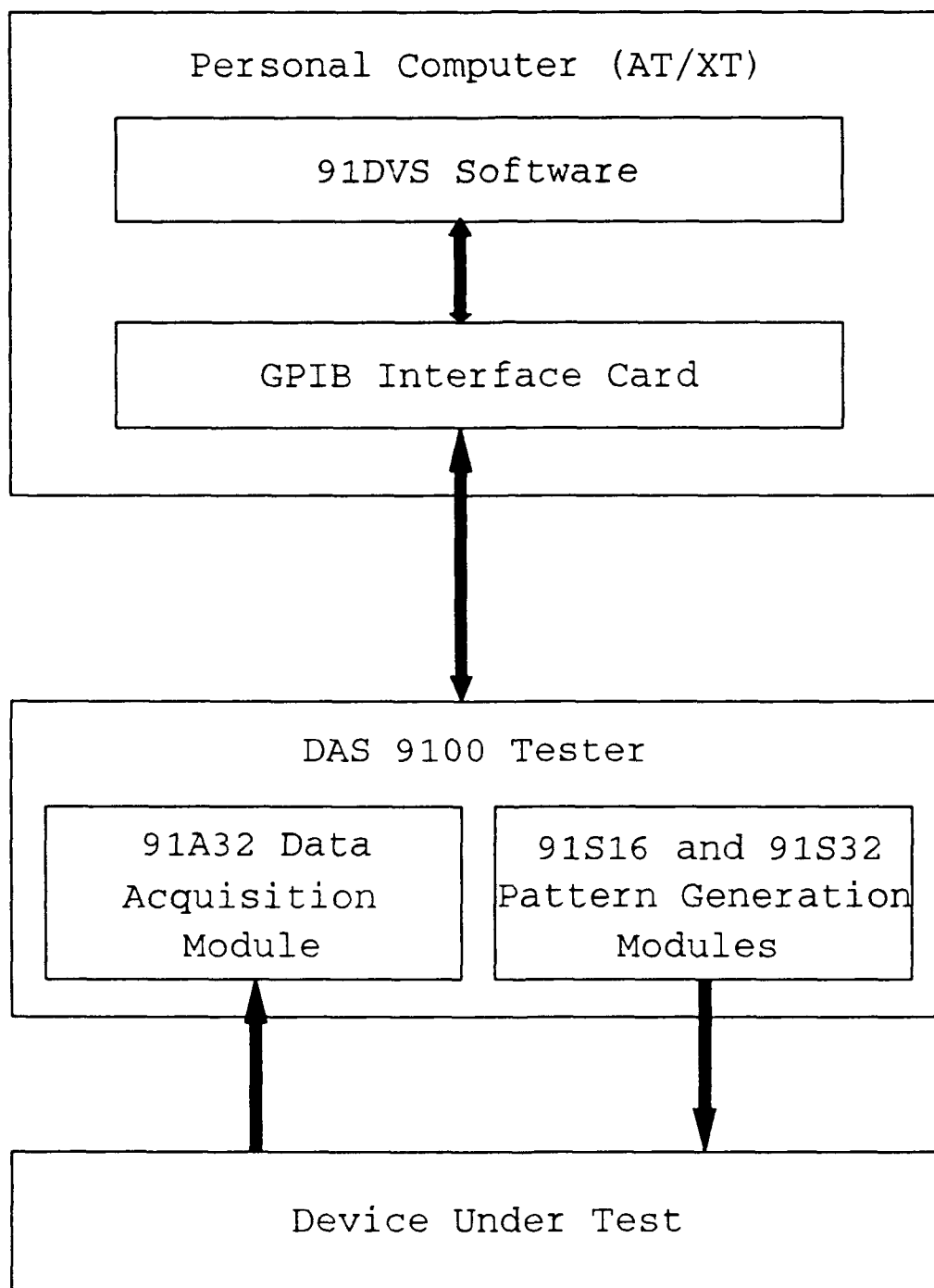
Figure 4.3   Functional Block Diagram of Integrated
             91DVS and DAS 9100 Test System

111

of 91S16 and 91S32 Pattern Generator modules for the DAS 9100 configuration at the NPS, the **DVS50** software program is presently installed on the controller PC. Use of the DVS50 software requires that the clock rates for the pattern generator and data acquisition modules of the DAS 9100 be the same [Ref. 24:p. 1-3]. Since the DAS 9100 configuration at the NPS uses a 91A32 Data Acquisition module with a 25 MHz maximum clock rate, this limits the maximum clock speed of the pattern generator modules to this same rate. The additional DVS50 requirement that data acquisition modules be clocked from the output of a pattern generator module limits the present NPS DAS 9100 configuration of pattern generator and data acquisition modules to 31 data acquisition channels and 47 pattern generation channels [Ref. 23:p. 73].

To use the DVS50 software to run a test two user supplied files must be available on the PC: the ".src" file and the ".das" file. The ".src" file is used by the DVS50 software to generate the configuration setup information used by the DAS 9100 during the test run. The ".das" test pattern file contains the sequence of input signal test vectors which will be applied to the DUT. An additional file which can be used with the DVS50 software is the ".sim" file. It contains the sequence of test vector inputs together with the expected output signal results and is used to compare actual test results against the expected results. The ".src", ".das", and ".sim" files must all be generated prior to initiating use of

the DVS50 program. Chapter 5 of Reference 24 contains specifications for the required formats of these files.

The MASM test vector files generated by the simulator and ATG processes in Genesil are <u>not</u> in the format of the ".das" and ".sim" files used by the DVS50 program. Additionally, the ".src" file must be generated at least once for each fabricated Genesil design which will be tested. To simplify the process of creating the files required by the DVS50 program, a conversion program, **convert.c**, was written in C to translate Genesil MASM test vector files to the ".das" and ".sim" formats. Additionally, if generation of a ".src" file is requested the conversion program will query the user on the needed information to produce a ".src" file. The source code for convert.c is contained in Appendix D. Both the source code and the executable program, convert.exe, are contained in the C:\DVSTEST subdirectory of the PC which has the DVS50 program installed on it.

To produce the user files required by the DVS50 program the Genesil MASM test vector files must first be exported from Genesil to a UNIX directory and then be transferred to the C:\DVSTEST subdirectory of the PC by using the KERMIT file transfer procedure. The conversion program can then be run. To obtain instructions on the use of the conversion program just enter the command "convert" from the C:\DVSTEST subdirectory.

An example of the ".src" file which can be produced using the conversion program is shown in Figure 4.4. Each pin on the DUT, as identified by name and pin number, shows whether a pattern generation (PAT) channel, data acquisition (ACQ) channel, or both will be attached to it. Additionally, the power supply pins have the label PS identified with them. The ".src" file also contains the details on the pattern generator module clocking rate (TIMEDEF information), pin threshold level (THRESHOLD information) and power supply pin definitions (PSDEF information) used by the DVS50 program to establish the proper setup information for the DAS 9100.

Examples of the ".das" and ".sim" files are provided in Figure 4.5 and Figure 4.6. They are nearly the same except that the ".sim" file contains the names and expected output sequence results for each output signal as well as the input signal names and test vectors. The ".das" file contains only the input signal names and test vector information.

Once ".src", ".das" and ".sim" files have been created through use of the conversion program testing on a chip via use of the DVS50 software can commence. Reference 23 contains a detailed tutorial on the use of this software. The following steps used to test the fabricated chips from the DFT_CHIP design illustrate the testing process.

1. Start the DVS50 program by typing DVS50 from within the C:\DVSTEST subdirectory which contains the ".src", "das" and ".sim" files.

2. From the DVS50 main menu choose **Compile Test Program** and provide the names of the ".src" and ".das" files. The

114

```
PROGRAM DFTCHIP;
PINDEF;
CLOCK2              : 20,    PAT,   ACQ;
CLOCK_SYS           : 13,    PAT,   ACQ;
DATCON              : 26,    PAT,   ACQ;
IN_PADS15           : 32,    PAT,   ACQ;
IN_PADS14           : 33,    PAT,   ACQ;
IN_PADS13           : 34,    PAT,   ACQ;
IN_PADS12           : 35,    PAT,   ACQ;
IN_PADS11           : 36,    PAT,   ACQ;
IN_PADS10           : 37,    PAT,   ACQ;
IN_PADS9            : 38,    PAT,   ACQ;
IN_PADS8            : 39,    PAT,   ACQ;
IN_PADS7            : 40,    PAT,   ACQ;
IN_PADS6            :  1,    PAT,   ACQ;
IN_PADS5            :  2,    PAT,   ACQ;
IN_PADS4            :  3,    PAT,   ACQ;
IN_PADS3            :  4,    PAT,   ACQ;
IN_PADS2            :  5,    PAT,   ACQ;
IN_PADS1            :  6,    PAT,   ACQ;
IN_PADS0            :  7,    PAT,   ACQ;
LOAD                : 24,    PAT,   ACQ;
M2                  : 23,    PAT;
M1                  : 22,    PAT;
MSKCON              : 27,    PAT,   ACQ;
OUTCON              : 19,    PAT;
REFCON              : 25,    PAT,   ACQ;
SERIAL_IN           : 31,    PAT,   ACQ;
SPCON               : 28,    PAT,   ACQ;
TESTIN              :  8,    PAT,   ACQ;
OUT_PADS4           : 18,    ACQ;
OUT_PADS3           : 17,    ACQ;
OUT_PADS2           : 16,    ACQ;
OUT_PADS1           : 15,    ACQ;
OUT_PADS0           : 14,    ACQ;
SHIFTOUT            : 21,    ACQ;
VDD_CORE            :  9,    PS 1;
VDD_RING            : 10,    PS 1;
VDD_CLOCK           : 11,    PS 1;
VSS_CLOCK           : 12,    PS 2;
VSS_RING            : 29,    PS 2;
VSS_CORE            : 30,    PS 2;
END;
TIMEDEF;
PAT  : ns 100;
END;
THRESHOLD;
ACQ  : TTL;
END;
PSDEF;
1   : mV 5000, mA 3000;
2   : mV 0;
END;
BEGIN;
END$
```

Figure 4.4 ".src" File Format

115

```
DFTCHIP
1
1
28
CLOCK2
CLOCK_SYS
DATCON
IN_PADS15
IN_PADS14
IN_PADS13
IN_PADS12
IN_PADS11
IN_PADS10
IN_PADS9
IN_PADS8
IN_PADS7
IN_PADS6
IN_PADS5
IN_PADS4
IN_PADS3
IN_PADS2
IN_PADS1
IN_PADS0
LOAD
M2
M1
MSKCON
OUTCON
REFCON
SERIAL_IN
SPCON
TESTIN
11010111010011110001010010000
00101110101111001111101010000
11110010101101110101111111101
00011111010011111100000001101
11100011011111000101100001100
00011010100000100110000110011
11101011011010011010001011101
00100111100001100111011111100
11110100100001001100000110000
00111000011110011001000010101
11000000001111110101111110000
00111110010110011001001100000
11101001111010100001000000001
00001100000011101011011111000
11100100000100111100110111110
00000111111000101001101111111
11100110101100010001111000010
00000111110001101111001100011
11001001110110110011000011000
00100000001101100101011111011
11010001111001000100001000000
00110110101110111101000110110
11010001110101011100001001001
00110000011100100110000001001
11100001100010001100000010011
00111111110011101111000111110
11000110110100111000011101010
```

Figure 4.5 ".das" File Format

116

```
DFTCHIP
1
1
34
CLOCK2
CLOCK_SYS
DATCON
IN_PADS15
IN_PADS14
IN_PADS13
IN_PADS12
IN_PADS11
IN_PADS10
IN_PADS9
IN_PADS8
IN_PADS7
IN_PADS6
IN_PADS5
IN_PADS4
IN_PADS3
IN_PADS2
IN_PADS1
IN_PADS0
LOAD
M2
M1
MSKCON
OUTCON
REFCON
SERIAL_IN
SPCON
TESTIN
OUT_PADS4
OUT_PADS3
OUT_PADS2
OUT_PADS1
OUT_PADS0
SHIFTOUT
11010111010011110001010001000......
00101110101111001111101010000......
11110010101101110101111111110100000.
00011111010011111100000011010100000.
11100011011111000101100001100000000
00011010100000100110001100110000000
11101011011010011010010111101010111
00100111100001100111011111100010111
11110100100001001100001100000101110
00111000011110011001000101011011110
11000000001111101011111000000000000
00111110010110011001001100000000000
11101001111010100001000000001000101
00001100000011101011011111000000101
11100100000010011100110111101010011
00000111111000101001101111111101001
11100110101100010001111000100010011
00000111110001101111001100001001011
11001001110110110011000010000000110
00100000001101100101011110110000110
11010001111001000100001000000010100
       dots (.) mean undetermined
       (i.e., not yet initialized)
```

Figure 4.6 ".sim" File Format

DVS50 program will use these files to produce the binary information files used to setup and control the DAS 9100 during the test run. Additionally, the **Channel Specification List**, which indicates how to connect the DAS 9100 probes to the DUT, is produced.

3. Return to the main DVS50 menus and Choose **Information.** From the Information menu select **Printing** to toggle the information output location to send information to the printer vice the screen. Next select **List Channel Specification** from the Information menu. This will cause a copy of the Channel Specification List to be printed out. This step needs to be done only for the first test on a chip design or if anything as been changed in the ".src" for subsequent tests.

4. Using the information from the Channel Specification List insert the DUT in the test jig and connect the DAS 9100 probes and power supply connectors. Figure 4.7 is an example of the Channel Specification List information used to connect up the DUT. Changes to the probe connections for subsequent tests and additional chips only needs to be done if the ".src" file PAT or ACQ channel information has been changed.

5. Reenter the DVS50 man menu and choose **Enter Test Menu** followed by **Run Test** to commence the test run. Ensure that the DAS 9100 has its power turned on and has completed its startup self-checks. When prompted by the DVS50 program turn on the power to the test jig for the DUT. The DVS50 program will then download the configuration setup and test vector pattern information to the DAS 9100. The DAS 9100 will run the test using this information and upload the results from the data acquisition channels to the PC. The test results are stored on the PC by the DVS50 software in a file with the extension name of ".A01". Upon completion of the test turn off the power supply to the test jig and reenter the main DVS50 menu.

6. To display the test results on the PC's screen select **Display Test Results** and provide the filenames of the ".A01" and ".sim" files. The DVS50 program will display the test results in timing diagram format with the ".A01" file actual test result information and ".sim" file expected test result information overlaid on each other in different colors for easy comparison. The screen display window can then be expanded, compressed, or moved through in a left or right direction using the PC function keys. The screen display window can display information on up to 24 signals at a time. Initially the display will show the alphabetically first 24 input

```
          9100 Device Verification Software - Version DVS50-2.31
          ***********************************************************

          Listing generated : ** 06/14/1990 ** 13:23:47 **
          Test Program in Use : DFTCHIP

               ***  CHANNEL SPECIFICATION LIST  ***

          ATTENTION :    Connect the  DAS-PODs to the pins of the  DUT
                         according to the following list.
                         Take care -    incorrect connections may cause
                         permanent damage to the DAS-PODs or the DUT !

          ***********************************************************
          *                                                         *
          * Trigger channel(s):          PG-POD         ACQ-POD     *
          *                                                         *
          *                              1B-STB-1        5D7-1       *
          *                                                         *
          ***********************************************************
          *                                                         *
          * ACQ clock channel(s):        PG-POD      EXTCLK-POD      *
          *                                                         *
          *                              1B-CLK-1       7C-CLK1-1    *
          *                                                         *
          ***********************************************************
          *                                                         *
          *  NR: NAME   PINNUMBER                 POD(S)             *
          *                                                         *
          *                        PG    ACQN   ACQF      other      *
          *   1  CLOCK2       20  1B7-1  5D6-1                       *
          *   2  CLOCK_SYS    13  1B6-1  5D5-1                       *
          *   3  DATCON       26  1B5-1  5D4-1                       *
          *   4  IN_PADS15    32  1B4-1  5D3-1                       *
          *   5  IN_PADS14    33  1B3-1  5D2-1                       *
          *   6  IN_PADS13    34  1B2-1  5D1-1                       *
          *   7  IN_PADS12    35  1B1-1  5D0-1                       *
          *   8  IN_PADS11    36  1B0-1  5C7-1                       *
          *   9  IN_PADS10    37  1A7-1  5C6-1                       *
          *  10  IN_PADS9     38  1A6-1  5C5-1                       *
          *  11  IN_PADS8     39  1A5-1  5C4-1                       *
          *  12  IN_PADS7     40  1A4-1  5C3-1                       *
          *  13  IN_PADS6      1  1A3-1  5C2-1                       *
          *  14  IN_PADS5      2  1A2-1  5C1-1                       *
          *  15  IN_PADS4      3  1A1-1  5C0-1                       *
          *  16  IN_PADS3      4  1A0-1  5B7-1                       *
          *  17  IN_PADS2      5  2D7-1  5B6-1                       *
          *  18  IN_PADS1      6  2D6-1  5B5-1                       *
          *  19  IN_PADS0      7  2D5-1  5B4-1                       *
          *  20  LOAD         24  2D4-1  5B3-1                       *
          *  21  M2           23  2D3-1                              *
          *  22  M1           22  2D2-1                              *
          *  23  MSKCON       27  2D1-1  5B2-1                       *
          *  24  OUTCON       19  2D0-1                              *
          *  25  REFCON       25  2C7-1  5B1-1                       *
          *  26  SERIAL_IN    31  2C6-1  5B0-1                       *
          *  27  SPCON        28  2C5-1  5A7-1                       *
          *  28  TESTIN        8  2C4-1  5A6-1                       *
          *  29  OUT_PADS4    18         5A5-1                       *
          *  30  OUT_PADS3    17         5A4-1                       *
          *  31  OUT_PADS2    16         5A3-1                       *
          *  32  OUT_PADS1    15         5A2-1                       *
          *  33  OUT_PADS0    14         5A1-1                       *
          *  34  SHIFTOUT     21         5A0-1                       *
          *  35  VDD_CORE      9                        PS1          *
          *  36  VDD_RING     10                        PS1          *
          *  37  VDD_CLOCK    11                        PS1          *
          *  38  VSS_CLOCK    12                        PS2          *
          *  39  VSS_RING     29                        PS2          *
          *  40  VSS_CORE     30                        PS2          *
          *                                                         *
          ***********************************************************
```

Figure 4.7   Channel Specification List Format

119

or output signals acquired during the test. To see other signals one of the signals presently displayed must be deleted before another signal can be inserted for display. Chapter 4 of Reference 24 provides specifics on user control of the DVS50 screen display. Figure 4.8 is an example from a test on the DFT_CHIP design of the screen display format which can be obtained for looking at test results.

Additional chips of the same design can be tested against the same set of test vectors by merely replacing the chip located in the test jig and then re-entering the DVS50 test menu (i.e., start with step five above).

2. Test Results for the DFT_CHIP Design

Using the steps enumerated in the previous subsection (Testing Methodology for the DFT_CHIP Design), the fabricated copies of the DFT_CHIP design were tested on the Das 9100 test gear. Each of the 12 fabricated chips were tested for overall manufacturing errors caused during fabrication by applying the test vector pattern for the maximum fault coverage test vector set obtained from the Genesil ATG process. Next, to validate that the DFT_CHIP design functionality was consistent with the desired operational logic the chips were tested using the test vector patterns produced from the Genesil simulation process. Each chip was tested using the test vectors produced from the test_parallel_in, test_serial_in, test_force and test_scan_-clock simulation check functions.

Since the logic design for all chips is the same and the purpose of the maximum fault coverage test vector set is to uncover manufacturing errors, the need to test the logic functionality of each chip in a commercial environment is not
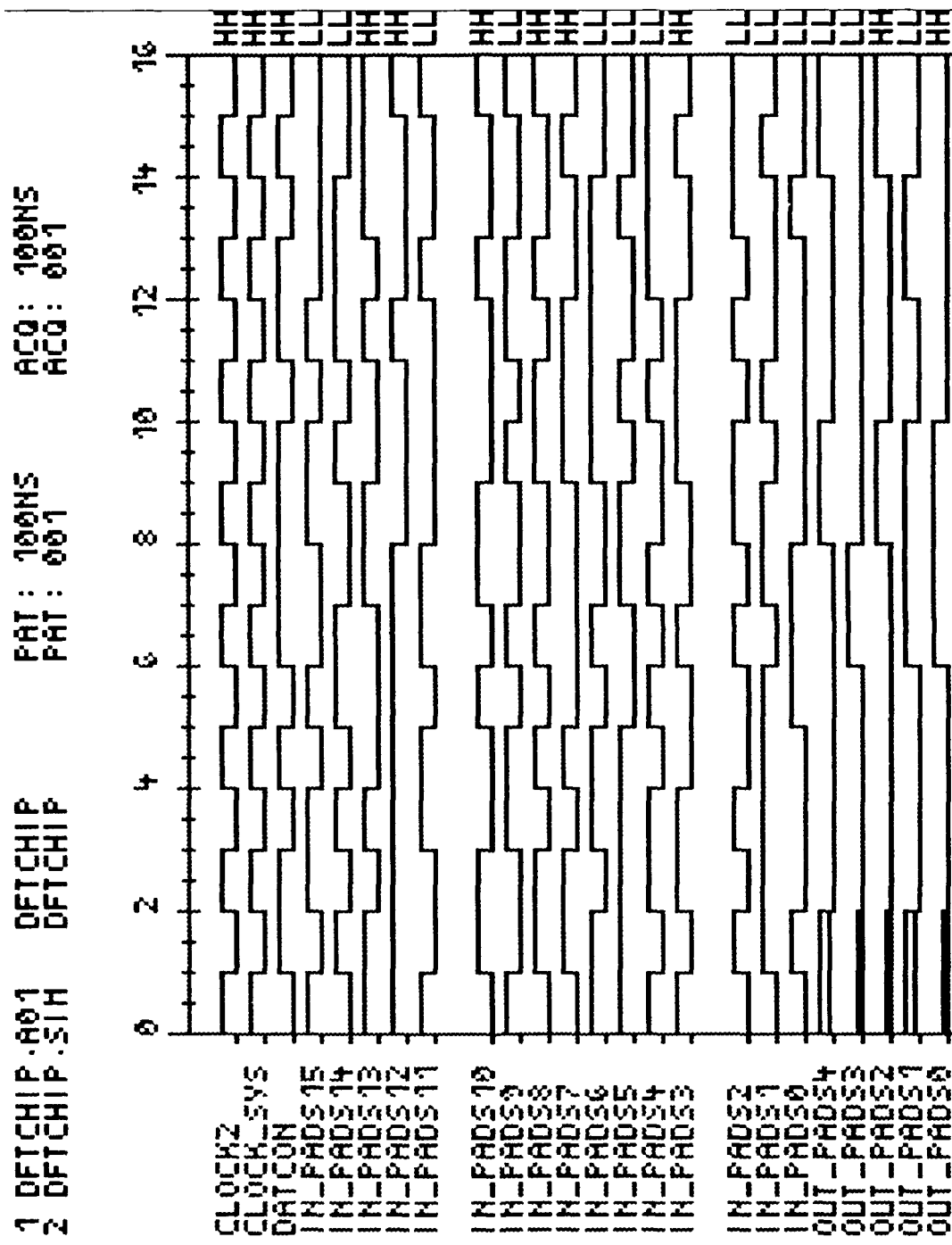
Figure 4.8   DVS50 Screen Display Format

present. Instead, if fault coverage is high, a commercial chip design may need to have only limited chip quantities checked for proper logic functionality. The remaining chips would then be accepted or rejected based only on the results obtained from testing with the maximum fault coverage test vector set.

For the DFT_CHIP design all copies of the fabricated chip passed all tests run on them. No unexpected logic functionality errors or manufacturing errors were observed. The DAS 9100 test gear, as controlled using the DVS50 software via the PC, made for a quick and easy means to conduct the tests and compare the results against the expected values determined by Genesil.

The final testing conducted on the fabricated chips was that done to try and determine the maximum clock speeds at which the two clock signals for the DFT_CHIP design could be applied. The global system and local scan path clock signals for the chip are generated via the test vector patterns which are applied. It takes a minimum of two consecutive test vectors to produce a complete clock cycle for either of these two signals. Since the DAS 9100 applies one test vector for each complete cycle of the pattern generator clock, the minimum time periods for the clock signals applied to the chip were twice the period of the pattern generation clock cycle used by the DAS 9100.

Using a 91A32 Data Acquisition module the DAS 9100 tester can have pattern generation clock cycle periods of 40ns, 50ns, 100ns, 200ns, 500ns, 1us, 2us, 5us, 10us, 20us, 50us, 100us, 200us, 500us, 1ms, 2ms and 5ms. Based on this, the maximum testable global system and local scan path clock speeds which could be applied to the DFT_CHIP design using the DAS 9100 configuration available at the NPS were 12.5 MHz. This same limitation would be found for any clock signal applied to a DUT that was derived from a Genesil produced test vector pattern.

First, the test vector sets which had both the global system and local scan path clocks cycling at the same speeds were applied to the chips. At clock speeds of 12.5 MHz and 10.0 MHz the chips would not produce the correct outputs. At a clock speed of 5.0 Mhz the outputs were correct.

Next, the test vectors produced from the test_scan_clock simulation function were applied. This function caused the global system clock signal to be first applied at a frequency half that of the local scan path clock signal and then to remain off while the local scan path clock continued to function. When this set of test vectors was run with the pattern generation clock frequency set such that the global system clock cycled at 6.25 MHz and the local scan path cycled at 12.5 MHz the outputs were as predicted.

These results tend to verify the predictions made by Genesil about the maximum operating frequencies for the two

123

clocks on the DFT_CHIP design. The maximum operating frequen-
cy for the global system clock is at least 5.0 MHz but is less
than 10.0 MHz. The exact maximum operating frequency for the
global system could not be determined since the DAS 9100 can
only change test frequencies in set increments. However, the
chips all performed better than the apparently conservative
Genesil estimate of 4.43 MHz for the global system clock. The
local scan path clock was able to run successfully run at 12.5
MHz which verified Genesil's prediction that it would run
faster then the global system clock. Higher speeds for the
local scan path clock could not be attempted due to the
maximum operating speed constraints of the present DAS 9100
configuration. Finally, these results validated the design
objective of being able to run the local scan path clock at a
higher frequency than that of the global system clock and to
continue scan path operations during periods that the global
system clock was not operating.

V. CONCLUSIONS

A. SUMMARY

This thesis has both described the benefits of including Design for Testability in a VLSI chip design and provided information on accomplishing this using the Genesil Silicon Compiler. Through a presentation of the methodology needed to implement a DFT design using Genesil, fabricate the design via MOSIS, and then test the chips on the DAS 9100 tester, a complete chip design production and testing sequence was illustrated.

The need for including DFT features in chip designs becomes increasingly important as the maximum fault coverage obtainable for more complex chips without DFT features decreases. The need for obtaining a high degree of fault coverage for VLSI chips was examined and a relationship between a chip's fault coverage, defect level and yield was developed. Both the Scan Path and Built-in Test techniques were discussed. They both provide a reasonable means of raising the fault coverage possible for a chip by providing greater controllability and observability of internal chip nodes. Whether one and/or both of these techniques should be used is decided by incorporating an evaluation of the specific design being produced.

125

Genesil, through its Testability Latch Blocks, provides a simple means of incorporating DFT into a chip design. Due to its ease of use, Genesil allows different DFT alternatives, in terms of techniques and/or feature placement, to be evaluated in a reasonable amount of time.

Including DFT features in a chip design can exact penalties in both the chip size and in performance characteristics such as operating speed and power consumption. Genesil designs are especially prone to experiencing penalties in chip size as object components are added. Proper floorplanning techniques, to include manual placement to locate objects, is critical in the size optimization effort for a Genesil produced design.

Genesil's simulation and automatic test generation features provide an integrated, easy to use means of developing test vectors which will either check the logic functionality of a design or provide the maximum possible fault coverage. Fault grading using the ATG process illustrated that test vector sets which provide good functional testing information may not provide a high degree of fault coverage. This again demonstrates that both categories of test vectors are needed for testing purposes. Being able to use an automated approach to developing test vectors provides a significant time savings. Without the simulation and ATG features of Genesil the evaluation of test vectors to be applied to the fabricated chips, on top of all the other

126

necessary design development steps, would not have been practical for a single person to accomplish.

Once Genesil designs are finished they are completely compatible with fabrication via MOSIS as long as a match is made between fabrication technologies. MOSIS provides excellent turn-around time service at a very reasonable cost of fabrication.

By utilizing the DVS50 software to control the DAS 9100 tester, chips fabricated from Genesil produced designs can be conveniently tested. The conversion program to translate test vectors from Genesil's MASM file format to the ".das" and ".sim" files used by the DVS50 software provides an easy means of utilizing any test vectors produced during the design process. Almost no knowledge of the DAS 9100 tester is needed to conduct tests if the DVS50 software is used. The DAS 9100 provides an adequate means of testing fabricated chips within the limitations of the maximum chip clock speeds which can be obtained.

Simulation results predicted by Genesil agreed with the results obtained during actual testing on the fabricated chip design. The use of the scan path for the design raised the fault coverage obtainable and lowered the number of test vectors needed to obtain maximum fault coverage as compared to the same design without a scan path. Obtaining expected test results for the maximum fault coverage test vector set indicated, to a high degree of confidence, that all the chips

were properly fabricated. Without the scan path feature the degree of confidence about the absence of manufacturing defects would have been smaller.

B. RECOMMENDATIONS

The following recommendations should be considered for implementation or additional investigation:

1. Genesil usage is <u>highly</u> CPU intensive. For the present setup of Genesil operating on the VAX a <u>large</u> amount of time is wasted waiting for Genesil to complete operations during periods of medium to high computer usage by other students. To greatly increase the speed of producing Genesil designs, Genesil should be moved to a platform which allows nonshared usage of a fast CPU.

2. The speed of applying test vectors on the DAS 9100 using the DVS50 software is presently constrained due to using a 91A32 Data Acquisition module which has its acquisition rate limited to 25 MHz. A 91A08 Data Acquisition module, which would raise this rate to a 50 MHz limit using the DVS50 software, should be acquired. This would allow an effective maximum rate of 25 MHz, vice the present 12.5 MHz, for chip clock signals generated via Genesil produced test vector sets.

3. Investigate the use of the Built-in Test DFT technique alone and or together with a scan path on a more complex Genesil produced VLSI design.

4. Investigate the incorporation of integrated DFT techniques and features into multiple chip VLSI designs to enhance complete board and or system level testing.

5. Genesil designs are presently limited by the need to utilize only Genesil library provided blocks during the design process. Investigate the means of incorporating optimized VLSI components, designed in a program like MAGIC, into Genesil designs to enhance the performance of critical chip components and to minimize overall chip size.

APPENDIX A. BASIC CHECK FUNCTIONS

This Appendix contains the GENIE language source code written for the basic building block check functions. These check functions only accomplish limited simulation tasks but may be grouped together or called from a higher level check function to run a complete simulation test.

```
func MSP (args value /* Function to set the mask register
        values in a parallel manner */
    sn SPCON 0
    sn IN_PADS @value
    sn MSKCON 1
    ck
    sn MSKCON 0
    }

func RSP (args value /* Function to set the reference
        register values in a parallel manner */
    sn SPCON 0
    sn IN_PADS @value
    sn REFCON 1
    ck
    sn REFCON 0
    }

func DSP (args value /* Function to set the data register
        values in a parallel manner */
    sn SPCON 0
    sn IN_PADS @value
    sn DATCON 1
    ck
    sn DATCON 0
    }


func TSS (args value /*Function to serially load 32 bits via
        the scanpath. Note: data is read in msb first. If
        the value used is not 32 bits in length 0's are read
        in to the left (msb's) of the loaded value */
    vars lsb length valstr i
    set valstr (bin @value)   /* Convert value into ascii
        string and assign to valstr */
```

```
set length (strlen @valstr) /* Determine length of
     valstr's ascii string */
sn M1 0 /* Insure testability latches set up for serial
        shifts*/
sn M2 0
if (@length != 33) {        /* String conversion appends an
        extra 0 onto the first position of the string so
        check for a length of 33 */
     for (i=0; @i<(33 - @length); ++i) { /* If value was
             not 16 bits then append the necessary zeroes
             to the front of the number before loading
             the msb */
        lsb = 0
        sn TESTIN @lsb
        ck
        )
     }
for (i=1; @i<@length; ++i) { /* Start with position 1
             (not 0) since the first bit in position 0 is
             the extra appended 0 which occurs during
             value to binary string conversion */
     lsb = (ord (substr @valstr @i 1 )) /* Extract the
             next msb from the string to shift in as
             data. Note: this line returns the ascii
             numeric value for this bit*/
     lsb = (@lsb - 48)      /* Convert ascii numeric value
             to the value to be shifted in next */
     sn TESTIN @lsb   /* Assign value to be shifted in to
                         the input pin */
     ck               /* clock in value */
     }
sn LOAD 0   /* Disable LOAD so forced value is not
             overwritten */
sn M1 1 /* Enable force operation for testability
        latches */
ck       /* Cycle to force all 32 bits into shift latch
         locations of testability latches */
sn M1 0 /* Return testability latches to normal shift
        operation */
sn LOAD 1   /* Return chip to normal ops on next clock
             cycle*/
}

func TS (args value1 value2  /* Function to serially load 1 to
        32 test bits via the scanpath. Upon completion of
        the scanpath serial load, a force operation is done
        on the testability latches to cause the values in
        the scanpath testability shift latches to be loaded
        into the testability data latches for propagation to
        the final output pins. Note: both the scan path
        shift operations and normal operations are restored
        upon completion. Note: data is read in msb first. If
```

```
          the value2 used is not value1 bits in length 0's are
          appended to the left (msb's) of value2. If value2
          has more bits than designated by value 1 then only
          the first value 1 msb's are shifted into the
          scanpath. */
vars lsb length valstr i
set valstr (bin @value2) /* Convert value into ascii
          string and assign to valstr */
set length (strlen @valstr)     /* Determine length of
          valstr's string */
sn M1 0 /*insure testability latches set up for serial
          shifts*/
sn M2 0
if (@length != (@value1 + 1)) {     /* String conversion
          appends an extra 0 onto the first position of
          the string so use value1 + 1   */
     for (i=0; @i<((@value1 + 1) - @length); ++i) {
          /* If value2 was value1 bits then append the
          necessary zeroes to the front of the number
          before loading the msb */
          lsb = 0
          sn TESTIN @lsb
          ck
          }
     }
for (i=1; @i<@length; ++i) {   /* Start with position 1
          (not 0) since the first bit in position 0 is the
          extra appended 0 which occurs during value to
          binary string conversion */
     lsb = (ord (substr @valstr @i 1 ))   /* Extract the
          next msb from the string to shift in as data.
          Note: this line returns the ascii numeric value
          for this bit*/
     lsb = (@lsb - 48)     /* Convert ascii numeric value
          to the value to be shifted in next */
     sn TESTIN @lsb  /* Assign value to be shifted in to
          the input pin */
     ck   /* clock in value */
     }
     force_in RB /* Force the values from the scanpath
          shift latches into the testability data latches
          for propagation to the output */
     }

func SS (args shift_type value1 value2
          /* General function for shifting in items serially.
          Arg 1 (shift_type) must be a string which specifies
          the items which will be shifted in. Allowable arg 1
          inputs are D (data), R (reference), M (mask), T
          (test via scanpath), DT (data and test), RT
          (reference and test), and MT (mask and test). Args 2
          to 3 are the number values which correspond to the
```

131

```
              strings to be shifted in. The order of these values
              must match the order of the items from arg 1. */
vars valstr1 length1 valstr2 length2 i
set valstr1 (bin @value1)        /* Convert value to ascii
              string */
set length1 (strlen @valstr1) /* Determine length of
              string */
set valstr2 (bin @value2)
set length2 (strlen @valstr2)
sn SPCON 1   /* Set SPCON for serial inputs */
if ((@length1 != 17) && (@value1 != null)) {
              /* If string length of input value is less than 17
              bits then cat zeros to the left until a full size
              string is present. Note: the bin process returns a
              string with an extra 0 appended to the left causing
              17-bit strings to be returned for a normal 16-bit
              value input. */
              for (i=0; @i<(17 - @length1); ++i) {
                   valstr1 = (cat 0 @valstr1)
                   }

              }
if ((@length2 != 17) && (@value2 != null)) {
              for (i=0; @i<(17 - @length2); ++i) {
                   valstr2 = (cat 0 @valstr2)
                   }

              }
for (i=1; @i<17; ++i) {   /* Loop to shift in data bits*/
              if (@shift_type == "D") {
                   LSB D @valstr1 @i      /*place next msb at serial
                        data input */
                   sn DATCON 1   /* Enable data to be shifted in */
                   ck              /* Cycle to shift in bit */
                   }

              }
for (i=1; @i<17; ++i) {   /* Shift in reference bits */
              if (@shift_type == "R") {
                   LSB R @valstr1 @i
                   sn REFCON 1
                   ck
                   }

              }
for (i=1; @i<17; ++i) {   /* Shift in mask bits */
              if (@shift_type == "M") {
                   LSB M @valstr1 @i
                   sn MSKCON 1
                   ck
                   }

              }
for (i=1; @i<17; ++i) {   /* shift in test bits */
              if (@shift_type == "T") {
                   LSB T @valstr1 @i
                   sn M1 0       /* Enable scanpath shift ops */
```

```
                        ck
                        }
                }
        for (i=1; @i<17; ++i) { /* Shift in data and test bits */
                if (@shift_type == "DT") {
                        LSB D @valstr1 @i
                        sn DATCON 1
                        LSB T @valstr2 @i
                        sn M1 0
                        ck
                        }
                }
        for (i=1; @i<17; ++i) { /* Shift in ref and test bits */
                if (@shift_type == "RT") {
                        LSB D @valstr1 @i
                        sn REFCON 1
                        LSB T @valstr2 @i
                        sn M1 0
                        ck
                        }
                }
        for (i=1; @i<17; ++i) { /* Shift in mask and test bits */
                if (@shift_type == "MT") {
                        LSB D @valstr1 @i
                        sn MSKCON 1
                        LSB T @valstr2 @i
                        sn M1 0
                        ck
                        }
                }
        sn DATCON 0 /* Disable data inputs */
        sn REFCON 0 /* Disable reference inputs */
        sn MSKCON 0 /* Disable mask inputs */
        sn M1 1      /* Disable scanpath shift ops */
        }

func LSB (args shift_type valstr_lsb i_lsb
        /* Function called by SS to assign the next bit of
        the input value string to the serial input pin */
        vars lsb
        lsb = (ord (substr @valstr_lsb @i_lsb 1))
                /* Obtain the next bit to be input as an ascii
                character */
        lsb = (@lsb - 48)    /* Convert ascii character to a
                number* /
        if ((@shift_type=="D") ¦ (@shift_type=="R") ¦
                (@shift_type=="M")) {
                sn SERIAL_IN @lsb    /* Assign next bit for to be
                        used for the serial input */
                }
        if (@shift_type == "T") {     /* Assign bit for test
                        shift in* /
```

133

```
            sn TESTIN @lsb
            }
    }


func force_in (args force_type
            /* Function to cause values which have been shifted
            in via the scanpath to be forced into the
            testability latch data latches. Note: once LOAD
            returns to 1 the normal propagated values from the
            data, ref, and mask registers reenter the data
            latches on the next clock cycle.  The RB option
            restores both shift and normal ops, RN restores
            only normal ops, and RS restores only shift ops. */
        sn LOAD 0   /* Disable normal inputs to testability
                        latch shift latches */
        sn M1 1 /* Enable force ops and disable scanpath shift
                ops*/
        sn M2 0
        ck  /* Cycle. This causes OUT_PADS to have an output
            based on the values shifted in via the scanpath.
            Note: no outputs will be obtained until 32 bits have
            been shifted into the scanpath to fill all spots
            along the serial scanpath testability latch line */
        if (@force_type=="RB") {
            sn M1 0 /* Restore normal shift operations */
            sn LOAD 1   /* Restore ops to normal on next clock
                        cycle */
            println "Both scan path shift ops and normal ops
                restored"
            }
        if (@force_type == "RN") {
            sn LOAD 1   /* Restore normal ops */
            println "Scan path shift ops disabled, normal ops
                restored"
            println "Restore shift ops by setting M1 to 0"
            }
        if (@force_type == "RS") {
            sn M1 0 /* Restore shift operations in scanpath */
            println "Scan path shift ops restored, normal ops
                disabled"
            println "Restore normal ops by setting LOAD to 1"
            }
    }


func swap_in (args swap_type
            /* Function to swap the data latch values with the
            testability latch shift latch values. Options for
            this function are RS to restore the scanpath shift
            ops but keep the normal ops disabled after
            completion, RN to restore the normal operations but
            keep the scan path shift ops disabled, and RB to
            restore both after completion. */
```

```
        sn LOAD 0      /* Disable normal input to data latches */
        sn M1 1        /* Set M1 and M2 for swap operation */
        sn M2 1
        ck             /* Cycle circuit to perform swap operation */
        if (@swap_type == "RB") {
            force_in RB /* A force operation must be done
                    immediately after the original data latch
                    contents move into the shift latch portion of
                    the testability latch */
            }
        if (@swap_type == "RN") {
            force_in RN
            }
        if (@swap_type == "RS") {
            force_in RS
            }
        }

func sample_in { /* Function to sample the data in the data
            latch and place it into the shift latch portion of
            the scanpath for scanpath output */
        sn LOAD 1
        sn M1 0
        sn M2 1
        ck
        sn M1 0
        sn M2 0
        sn LOAD 1
        }

func tog { /* Function to set up toggle patterns for clocks */
        toggle CLOCK 1 '(0 10 20)       /* Set up toggle scheme for
            CLOCK signal */
        tag CLOCK cycle none        /* Reset CLOCK cycle tags */
        tag CLOCK cycle falling  /* Tag CLOCK so that "ck"
            command advances to the next rising edge of the
            CLOCK signal */
        tag CLOCK step both
        toggle CLOCK2 0 '(5 15 25) /* Set up toggle scheme for
            CLOCK2 signal */
        tag CLOCK2 cycle none /* Do not base "ck" commands on
            state of CLOCK2 */
        tag CLOCK2 step both /* Needed to get step resolution */
        }

func untog { /* Function to untoggle clocks - must be used
            before running test vectors created by the traceobj
            command */
        untoggle CLOCK
        untoggle CLOCK2
        }
```

APPENDIX B.   HIGH LEVEL CHECK FUNCTIONS

This Appendix contains the GENIE language source code
written for the high level simulation check functions. These
check functions use the basic level check functions to
automatically accomplish a complete simulation test. Use of
the Genesil traceobj command also causes these check functions
to produce MASM test vector files.

```
func initall ( /* Function to intialize all input pins at time
          -1. Note: all pins must have an initialization value
          assigned for the simulation process to work
          correctly. Also, note: prior to running this
          function the time should be reset to time -1 by
          using the command inittoggles. */
     sn TESTIN 0 /* Set scanpath input to 0 */
     sn LOAD 1    /* Enable test latches to move values from
                  combiner 1 to  combiner 2 */
     sn M1 0      /* Set scanpath test latch operation for
                  shift in/out operation */
     sn M2 0
     sn SERIAL_IN 0   /* Set serial input pin to value of 0 */
     sn REFCON 1      /* Allow reference values to be input */
     sn OUTCON 1      /* Enable outputs to OUT_PADS */
     sn SPCON 0       /* Setup for parallel inputs*/
     sn DATCON 1      /* Allow data values to be input */
     sn MSKCON 1      /* Allow mask values to be input */
     sn IN_PADS 0X0000 /* Set all parallel input pins to 0 */
     ck  /* Cycle to bring time from -1 to 0  and input
          0X0000 on the mask reference and data registers */
     ck  /* Cycle to propagate values from registers to the
          data latches the testability latches */
     sample_in   /* Initialize scanpath nodes by sample
                  operation*/
        sn MSKCON 0  /* Disable mask value input */
        sn REFCON 0  /* Disable reference value input */
        sn DATCON 0  /* Disable data value input */
        ck           /* Cycle to pass results to output pins */
     }
```

```
func test_parallel_in {    /* Function to produce test vectors
        which test the propering functioning of all portions
        of the chip except the testability latches but which
        emphasize the parallel load operations of the data,
        mask, and reference registers */
    inittoggles
    untog    /* Untoggle default clock toggle definitions */
    tog      /* Toggle clock toggle definitions */
    traceobj vecspara / /* Initiate traceobj command to put
        test results in a file named vecspara */
    initall /* Initialize chip */

    /* initialize mask and reference registers to 0XFFFF */
    MSP 0XFFFF
    RSP 0XFFFF

    /* Load various input values in a parallel manner into
    the data register to check the chip's operation.   The
    inputs cause DATAOUT values to range from 00 to 10. */
    DSP 0X0000   /* DATAOUT = 00 */
    DSP 0X0001   /* DATAOUT = 01 */
    DSP 0X0012   /* DATAOUT = 02 */
    DSP 0X0122   /* DATAOUT = 03 */
    DSP 0X0123   /* DATAOUT = 04 */
    DSP 0X1234   /* DATAOUT = 05 */
    DSP 0X2345   /* DATAOUT = 06 */
    DSP 0X3456   /* DATAOUT = 07 */
    DSP 0X4567   /* DATAOUT = 08 */
    DSP 0X5678   /* DATAOUT = 08 */
    DSP 0X6789   /* DATAOUT = 08 */
    DSP 0X789A   /* DATAOUT = 08 */
    DSP 0X89AB   /* DATAOUT = 07 */
    DSP 0X9ABC   /* DATAOUT = 08 */
    DSP 0XABCD   /* DATAOUT = 09 */
    DSP 0XBCDE   /* DATAOUT = 0A */
    DSP 0XCDEE   /* DATAOUT = 0B */
    DSP 0XCDEF   /* DATAOUT = 0C */
    DSP 0XDEEF   /* DATAOUT = 0D */
    DSP 0XEEFF   /* DATAOUT = 0E */
    DSP 0XEFFF   /* DATAOUT = 0F */
    DSP 0XFFFF   /* DATAOUT = 10 */

    /* Load various input values in a parallel manner into
    the reference register to check the chip's operation.
    The inputs cause DATAOUT values to range from 00 to 10.
    */
    RSP 0X0000   /* DATAOUT = 00 */
    RSP 0X0001   /* DATAOUT = 01 */
    RSP 0X0012   /* DATAOUT = 02 */
    RSP 0X0122   /* DATAOUT = 03 */
    RSP 0X0123   /* DATAOUT = 04 */
    RSP 0X1234   /* DATAOUT = 05 */
```

```
RSP 0X2345   /* DATAOUT = 06 */
RSP 0X3456   /* DATAOUT = 07 */
RSP 0X4567   /* DATAOUT = 08 */
RSP 0X5678   /* DATAOUT = 08 */
RSP 0X6789   /* DATAOUT = 08 */
RSP 0X789A   /* DATAOUT = 08 */
RSP 0X89AB   /* DATAOUT = 07 */
RSP 0X9ABC   /* DATAOUT = 08 */
RSP 0XABCD   /* DATAOUT = 09 */
RSP 0XBCDE   /* DATAOUT = 0A */
RSP 0XCDEE   /* DATAOUT = 0B */
RSP 0XCDEF   /* DATAOUT = 0C */
RSP 0XDEEF   /* DATAOUT = 0D */
RSP 0XEEFF   /* DATAOUT = 0E */
RSP 0XEFFF   /* DATAOUT = 0F */
RSP 0XFFFF   /* DATAOUT = 10 */

/* Load various input values in a parallel manner into
the mask register to check the chip's operation.  The
inputs cause DATAOUT values to range from 00 to 10. */
MSP 0X0000   /* DATAOUT = 00 */
MSP 0X0001   /* DATAOUT = 01 */
MSP 0X0012   /* DATAOUT = 02 */
MSP 0X0122   /* DATAOUT = 03 */
MSP 0X0123   /* DATAOUT = 04 */
MSP 0X1234   /* DATAOUT = 05 */
MSP 0X2345   /* DATAOUT = 06 */
MSP 0X3456   /* DATAOUT = 07 */
MSP 0X4567   /* DATAOUT = 08 */
MSP 0X5678   /* DATAOUT = 08 */
MSP 0X6789   /* DATAOUT = 08 */
MSP 0X789A   /* DATAOUT = 08 */
MSP 0X89AB   /* DATAOUT = 07 */
MSP 0X9ABC   /* DATAOUT = 08 */
MSP 0XABCD   /* DATAOUT = 09 */
MSP 0XBCDE   /* DATAOUT = 0A */
MSP 0XCDEE   /* DATAOUT = 0B */
MSP 0XCDEF   /* DATAOUT = 0C */
MSP 0XDEEF   /* DATAOUT = 0D */
MSP 0XEEFF   /* DATAOUT = 0E */
MSP 0XEFFF   /* DATAOUT = 0F */
MSP 0XFFFF   /* DATAOUT = 10 */
untraceobj   /* Close vecspara file */
}


func test_serial_in {/* Function to test the serial loading
        capabilities of the chip. Values are loaded into the
        data, reference and mask registers and into the
        testability latches via use of the SS function. */
    inittoggles
    untog
```

```
tog
traceobj vecsserl / /* Open file for test vector
     results*/
initall /* Initialize chip */

/* Initialize mask and reference registers and testability
latch scan path latches to all ones */
     SS MT 0XFFFF 0XFFFF
     SS RT 0XFFFF 0XFFFF

/* Load values in a serial mannert into data register  to
check for proper chip operation. Also include some new inputs
to the scan path to change the values scanned out of the chip.
The comments indicate the expected output values after the
completion of each operation. The println commands serve
as an example of a method of checking the simulation results
as the check function progresses. */
     SS DT 0X0000 0X0000 /* DATAOUT = 00 SHIFTOUT STILL = 1*/
     ck /* cycle to obtain expected output from scan path */
     println "DT 0X0000 0X0000 done, 00 1 expected output,
          current time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
     SS DT 0X1000 0X0000 /* DATAOUT = 01 SHIFTOUT NOW = 0 */
     ck
     println "DT 0X1000 0X0000 done, 01 0 expected output,
          current time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
     SS DT 0X2100 0XFFFF /* DATAOUT = 02 SHIFTOUT STILL = 0 */
     ck
     println "DT 0X2100 0XFFFF done, 02 0 expected output,
          current time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
     SS DT 0X2210 0XFFFF /* DATAOUT = 03 SHIFTOUT NOW = 1 */
     ck
     println "DT 0X2210 0XFFFF done, 03 1 expected output,
          current time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
     SS D 0X3210 /* DATAOUT = 04 */
     ck
     println "D 0X3210 done, 04 1 expected output, current
          time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
     SS D 0X4321 /* DATAOUT = 05 */
     ck
     println "D 0X4321 done, 05 1 expected output, current
          time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
     SS D 0X5432 /* DATAOUT = 06 */
     ck
     println "D 0X5432 done, 06 1 expected output, current
          time" (gettime)
     println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
```

```
SS D 0X6543 /* DATAOUT = 07 */
ck
println "D 0X6543 done, 07 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XA987 /* DATAOUT = 08 */
ck
println "D 0XA987 done, 08 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XBAA9 /* DATAOUT = 09 */
ck
println "D 0XBAA9 done, 09 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XEDCA /* DATAOUT = 0A */
ck
println "D 0XEDCA done, 0A 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XEEDC /* DATAOUT = 0B */
ck
println "D 0XEEDC done, 0B 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XFEDC /* DATAOUT = 0C */
ck
println "D 0XFEDC done, 0C 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XFEED /* DATAOUT = 0D */
ck
println "D 0XFEED done, 0D 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XFFEE /* DATAOUT = 0E */
ck
println "D 0XFFEE done, 0E 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XFFFE /* DATAOUT = 0F */
ck
println "D 0XFFFE done, 0F 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
SS D 0XFFFF    /* DATAOUT = 10 */
ck
println "D 0XFFFF done, 10 1 expected output, current
     time" (gettime)
println "actual values" (snb OUT_PADS) (snb SHIFTOUT)
untraceobj
)
```

```
func test_force {      /* Function to check the serial load
          and force operations of the testability latches and
          to also check the adder and final output results for
          cases where the final output ranges from 10 hex to
          1C hex (ie output due to force operations involves
          values which can not be obtained through the inputs
          to the correlator). */

     /* Initialize time, and clock toggle definitions */
     inittoggles
     untog
     tog
     traceobj vecstfor / /* Start tracing of operations and
          send output to file vecstfor.SMO */
     initall  /* Initialize chip */

     /* cause output to be 10 hex */
     MSP 0XFFFF
     RSP 0XFFFF
     DSP 0XFFFF
     sample_in    /* Cause shift latches in testability latches
          to hold values which causes an output of 10hex */
     TS 16 0XFDFF /* Cause output of 11 */
     TS 16 0XFDFF /* Cause output of 12 */
     TS 16 0XFFFF /* Cause output of 13 */
     TS 16 0XFFFF /* Cause output of 14 */
     TS 16 0XF5FF /* Cause output of 15 */
     TS 16 0XF5FF /* Cause output of 16 */
     TS 16 0XF9F3 /* Cause output of 17 */
     TS 16 0XF9F3 /* Cause output of 18 */
     TS 16 0XF40F /* Cause output of 19 */
     TS 16 0XF40F /* Cause output of 1A */
     TS 16 0X6123 /* Cause output of 1B */
     TS 16 0X6123 /* Cause output of 1C */
     untraceobj    /* Close vecstfor file */
     }


func test_scan_clock {    /* Function to test the ability of
          the scan path clock (CLOCK2) to operate at a faster
          speed than the global system clock (CLOCK2), and
          also to test the ability to operate the scan path
          with the global system clock off */
     vars i /* variable for the loop */
     inittoggles
     untog

     /* Provide toggle definitions for clocks */
     toggle CLOCK2 0 '(0 5 10)
     /* The following unwieldy looking toggle definition for
     the CLOCK signal will cause it to operate at the same
```

speed as the CLOCK2 signal during the initialization
process, at half the speed as the CLOCK2 signal for
the first part of the test, and then cause the CLOCK
signal to remain at zero for the remainder of the test */
toggle CLOCK 0 '(0 5 10 15 20 25 30 35 40 50 60 70 80 90
    100 110 120 130 140 150 160 170 180 190 200 210 220
    230 240 250 260 270 280 290 300 310 320 330 340 350
    360 370 380 390 400 410 420 430 440 450 460 470
    1000)
tag CLOCK2 step both

traceobj vecssclk /
initall

/* Put initial values into data, mask, and reference
registers to start the test. Note: each command must be
done twice since the function calls have clock cycles
based on CLOCK2 now, but you still need the same number
of clock cycles as normal to occur with the CLOCK signal
to load the values into the registers. */
MSP 0XFFFF
MSP 0XFFFF
RSP 0XFFFF
RSP 0XFFFF
DSP 0XCCCC   /* Loading this value into the data
    register will cause a value of 2448 2448 (hex) to be
    loaded into the scan path when a swap operation is
    done. The sequence of ones and zeros coming out of
    the scan path can then be checked against these
    values to verify proper operation of the chip */
DSP 0XCCCC

swap_in RS /* swap 2448 2448 into the scan path */
for(i=1; @i<33; ++i) { /* scan out the value 2448 2448
        with the CLOCK signal operating */
    ck /* cycle the CLOCK2 signal */
    }

sn LOAD 1
ck 2 /* reload data latch of testability latches */
swap_in RS   /* reload the scan path with the same
            values as above */
for(i=1; @i<33; ++i) { /* scan values out with system
    clock off to validate the ability to operate the DFT
    scan path feature with no system clock functioning
    as might be done for in-site testing */
    ck
    }
untraceobj
}

APPENDIX C. TIMING ANALYSIS REPORTS

This Appendix provides the timing analysis reports generated for the global system and local scan path clock signals in the DFT_CHIP design. A Clock, Setup and Hold and Violations report was generated for each of these two clock signals. This information was used to verify that the DFT_CHIP design did not have any timing relationship conflicts and also to determine an anticipated maximum operating speed for both the global system and local scan path clock signals.

```
****************************************************************
        Genesil Version v7.1 -- Thu Jun  7 15:55:45 1990
Chip: ˉgenpooler/pooler/DFT_CHIP
    Timing Analyzer
****************************************************************
CLOCK REPORT MODE
----------------------------------------------------------------
Fabline: VTI_CN20A                      Corner: GUARANTEED
    Junction Temperature:75 deg C       Voltage:5.00v
    External Clock: CLOCK
  Included setup files:
    #0  CLOCK_defaults        (CLOCK 5V 75 degree Cent results)
----------------------------------------------------------------
                    CLOCK TIMES (minimum)
Phase 1 High:   113.0 ns            Phase 2 High:     40.0 ns

Cycle (from Ph1): 36.2 ns     Cycle (from Ph2):    146.9 ns

Minimum Cycle Time: 152.9 ns  Symmetric Cycle Time: 225.9 ns

----------------------------------------------------------------
                    CLOCK WORST CASE PATHS
Minimum Phase 1 high time is    113.0    ns set by:
    ** Clock delay: 7.2ns (120.1-113.0)
        Node                  Cumulative Delay      Transition
        dataout[4]/(internal)      120.1                rise
        dataout[4]/DOUT            113.2                fall
        output/cout[4]             113.1                fall
        output/cout[4]'            109.9                fall
```

	Cumulative Delay	Transition
output/out[4]	106.7	fall
adder/out[4]	106.7	fall
adder/out[4]'	106.5	fall
adder/c3	87.4	fall
adder/c3'	87.3	fall
adder/c2out[0]	63.5	fall
combiner2/c2out[0]'	63.5	fall
combiner2/c2out[0]	61.0	fall
combiner2/x13	57.1	fall
tlatch32/x13	57.1	fall
tlatch32/x13'	56.3	fall
tlatch32/t13	46.7	fall
combiner1/t13	46.7	fall
combiner1/t13'	44.9	fall
combiner1/t10	39.4	rise
combiner1/t10'	37.3	rise
combiner1/xout[7]	30.0	fall
xnorreg/xout[7]	30.0	fall
xnorreg/xout[7]'	28.5	fall
xnorreg/x[7]	25.3	fall
xnorreg/rfout[7]	21.0	fall
ref_in/ref1/rfout[7	21.0	fall
ref_in/ref1/rfout[7	19.1	fall
ref_in/ref1/phase_a	10.2	rise
clock/phase_a	10.1	rise
CLOCK	0.0	rise

Minimum Phase 2 high time is   40.0   ns set by:
** Clock delay: 6.1ns (46.0-40.0)

Node	Cumulative Delay	Transition
maskin/mask1/(internal)	46.0	rise
mask_in/mask1/mout[0]	41.1	fall
mask_in/mask1/mout[0]'	41.0	fall
mask_in/mask1/sp_con	35.4	fall
spcon/sp_con	35.1	fall
spcon/sp_con'	15.3	fall
spcon/phase_b	9.5	rise
clock/phase_b	9.4	rise
CLOCK	0.0	fall

Minimum cycle time (from Ph1) is   36.2   ns set by:
** Clock delay: 2.5ns (38.7-36.2)

Node	Cumulative Delay	Transition
mask_in/mask1/(internal)	38.7	rise
mask_in/mask1/mout[7]	37.3	fall
mask_in/mask1/mout[7]'	37.3	fall
mask_in/mask1/sp_con	31.7	fall
spcon/sp_con	31.4	fall
spcon/sp_con'	11.5	fall
*spcon/(internal)	7.3	rise
SPCON	0.0	fall

144

```
Minimum cycle time (from Ph2) is   146.9     ns set by:
   ** Clock delay: 1.1ns (148.0-146.9)
      Node                      Cumulative Delay        Transition
      dataout[4]/(internal)         148.0                 rise
      dataout[4]/DOUT               147.1                 fall
      output/cout[4]                147.1                 fall
      output/cout[4]'               143.9                 fall
      output/out[4]                 140.6                 fall
      adder/out[4]                  140.6                 fall
      adder/out[4]'                 140.5                 fall
      adder/c3                      121.3                 fall
      adder/c3'                     121.2                 fall
      adder/c2out[0]                 97.5                 fall
      combiner2/c2out[0]             97.4                 fall
      combiner2/c2out[0]'            94.9                 fall
      combiner2/x13                  91.0                 fall
      tlatch32/x13                   91.0                 fall
      tlatch32/x13'                  90.2                 fall
      tlatch32/t13                   80.7                 fall
      combiner1/t13                  80.7                 fall
      combiner1/t13'                 78.8                 fall
      combiner1/t10                  73.3                 rise
      combiner1/t10'                 71.2                 rise
      combiner1/xout[7]              63.9                 fall
      xnorreg/xout[7]                63.9                 fall
      xnorreg/xout[7]'               62.4                 fall
      xnorreg/x[7]                   59.2                 fall
      xnorreg/rfout[7]               55.0                 fall
      ref_in/ref1/rfout[7]           55.0                 fall
      ref_in/ref1/rfout[7]'          53.0                 fall
      *ref_in/ref1/(internal)        46.9                 fall
      ref_in/ref1/rout[7]            40.9                 fall
      ref_in/ref1/rout[7]'           40.9                 fall
      ref_in/ref1/sp_con             35.3                 fall
      spcon/sp_con                   35.1                 fall
      spcon/sp_con'                  15.3                 fall
      spcon/phase_b                   9.5                 rise
      clock/phase_b                   9.4                 rise
      CLOCK                           0.0                 fall
   -------------------------------------------------------------


   ************************************************************
        Genesil Version v7.1 -- Thu Jun  7 15:54:35 1990
   Chip: ˉgenpooler/pooler/DFT_CHIP   Timing Analyzer
   ************************************************************
   SETUP AND HOLD MODE
   -------------------------------------------------------------
   Fabline: VTI_CN20A                    Corner: GUARANTEED
     Junction Temperature:75 deg C       Voltage:5.00v
     External Clock: CLOCK
    Included setup files:
     #0  CLOCK_defaults        (CLOCK 5V 75 degree Cent results)
```

```
-----------------------------------------------------------
                INPUT SETUP AND HOLD TIMES (ns)
Input               Setup Time           Hold Time
                 Ph1(f)    Ph2(f)      Ph1(f)    Ph2(f)
DATCON            ---       22.8         ---       1.2      PATH
IN_PADS[0]        ---       18.7         ---       1.1      PATH
IN_PADS[10]       ---       17.4         ---       1.2      PATH
IN_PADS[11]       ---       17.3         ---       1.2      PATH
IN_PADS[12]       ---       17.8         ---       1.2      PATH
IN_PADS[13]       ---       18.0         ---       1.2      PATH
IN_PADS[14]       ---       17.8         ---       1.2      PATH
IN_PADS[15]       ---       17.6         ---       1.2      PATH
IN_PADS[1]        ---       18.3         ---       1.1      PATH
IN_PADS[2]        ---       18.1         ---       1.1      PATH
IN_PADS[3]        ---       17.7         ---       1.1      PATH
IN_PADS[4]        ---       17.2         ---       1.2      PATH
IN_PADS[5]        ---       17.0         ---       1.2      PATH
IN_PADS[6]        ---       16.8         ---       1.2      PATH
IN_PADS[7]        ---       16.8         ---       1.2      PATH
IN_PADS[8]        ---       17.9         ---       1.2      PATH
IN_PADS[9]        ---       18.0         ---       1.2      PATH
LOAD              ---       10.7         ---       1.2      PATH
M1                ---        3.9         ---       1.2      PATH
M2                ---        3.9         ---       1.2      PATH
MSKCON            ---       23.6         ---       1.2      PATH
OUTCON            ---        3.9         ---       1.2      PATH
REFCON            ---       22.5         ---       1.2      PATH
SERIAL_IN         ---       16.5         ---       1.2      PATH
SPCON             ---       36.2         ---       1.2      PATH
TESTIN            ---       ---          ---       ---      PATH
```

```
**********************************************************
      Genesil Version v7.1 -- Thu Jun  7 15:56:35 1990
Chip: ¯genpooler/pooler/DFT_CHIP
    Timing Analyzer
**********************************************************
VIOLATION MODE
-----------------------------------------------------------
Fabline: VTI_CN20A                   Corner: GUARANTEED
  Junction Temperature:75 deg C      Voltage:5.00v
  External Clock: CLOCK
 Included setup files:
 #0  CLOCK_defaults        (CLOCK 5V 75 degree Cent results)
-----------------------------------------------------------
                    NO VIOLATIONS
  Hold time check margin: 4.0ns
-----------------------------------------------------------
```

```
**********************************************************
         Genesil Version v7.1 -- Thu Jun  7 17:31:18 1990
Chip: ~genpooler/pooler/DFT_CHIP
    Timing Analyzer
**********************************************************
CLOCK REPORT MODE
-----------------------------------------------------------
Fabline: VTI_CN20A                     Corner: GUARANTEED
  Junction Temperature:75 deg C        Voltage:5.00v
  External Clock: CLOCK2
 Included setup files:
  #0  CLOCK2_defaults      (CLOCK2 5V 75 degree Cent results)
-----------------------------------------------------------
                    CLOCK TIMES (minimum)
Phase 1 High:   17.5 ns         Phase 2 High:    6.6 ns

Cycle (from Ph1):   25.8 ns     Cycle (from Ph2):   24.7 ns

Minimum Cycle Time:  25.8 ns   Symmetric Cycle Time:  35.0 ns
-----------------------------------------------------------
                    CLOCK WORST CASE PATHS
Minimum Phase 1 high time is     17.5 ns set by:
  ** Clock delay: 9.2ns (26.7-17.5)
     Node                Cumulative Delay      Transition
     tlatch32/(internal)      26.7                 fall
     tlatch32/x12             24.2                 fall
     tlatch32/x12'            23.4                 fall
     tlatch32/phase_ta         8.7                 rise
     clock2/phase_ta           8.7                 rise
     CLOCK2                    0.0                 rise

Minimum Phase 2 high time is      6.6 ns set by:
  ** Clock delay: 8.7ns (15.4-6.6)
     Node                Cumulative Delay      Transition
     tlatch32/(internal)      15.4                 fall
     tlatch32/phase_tb         7.7                 rise
     clock2/phase_tb           7.7                 rise
     CLOCK2                    0.0                 fall

Minimum cycle time (from Ph1) is    25.8 ns set by:
  ** Clock delay: 8.7ns (34.5-25.8)
     Node                Cumulative Delay      Transition
     tout/(internal)          34.5                 fall
     tout/testout             34.1                 rise
     tlatch32/testou          34.0                 rise
     tlatch32/testou          32.6                 rise
     *tlatch32/(intenal)      27.6                 fall
     tlatch32/x14             22.0                 rise
     tlatch32/x14'            21.8                 rise
     tlatch32/phase_ta         8.7                 rise
     clock2/phase_ta           8.7                 rise
     CLOCK2                    0.0                 rise


                           147
```

```
Minimum cycle time (from Ph2) is    24.7 ns set by:
  ** Clock delay: 6.7ns (31.4-24.7)
    Node                    Cumulative Delay        Transition
    tlatch32/x14                  31.4               fall
    tlatch32/x14'                 31.3               fall
    *tlatch32/(internal)          23.0               rise
    tlatch32/testout              21.7               fall
    tlatch32/testout'             20.3               fall
    tlatch32/phase_tb              7.7               rise
    clock2/phase_tb                7.7               rise
    CLOCK2                         0.0               fall
-----------------------------------------------------------------


*****************************************************************
       Genesil Version v7.1 -- Thu Jun  7 17:30:24 1990
Chip: ˜genpooler/pooler/DFT_CHIP
   Timing Analyzer
*****************************************************************
SETUP AND HOLD MODE
-----------------------------------------------------------------
Fabline: VTI_CN20A                      Corner: GUARANTEED
  Junction Temperature:75 deg C         Voltage:5.00v
  External Clock: CLOCK2
 Included setup files:
  #0  CLOCK2_defaults       (CLOCK2 5V 75 degree Cent results)
-----------------------------------------------------------------
                  INPUT SETUP AND HOLD TIMES (ns)
Input              Setup Time            Hold Time
                Ph1(f)    Ph2(f)      Ph1(f)    Ph2(f)
DATCON           ---       ---         ---       ---      PATH
IN_PADS[0]       ---       ---         ---       ---      PATH
IN_PADS[10]      ---       ---         ---       ---      PATH
IN_PADS[11]      ---       ---         ---       ---      PATH
IN_PADS[12]      ---       ---         ---       ---      PATH
IN_PADS[13]      ---       ---         ---       ---      PATH
IN_PADS[14]      ---       ---         ---       ---      PATH
IN_PADS[15]      ---       ---         ---       ---      PATH
IN_PADS[1]       ---       ---         ---       ---      PATH
IN_PADS[2]       ---       ---         ---       ---      PATH
IN_PADS[3]       ---       ---         ---       ---      PATH
IN_PADS[4]       ---       ---         ---       ---      PATH
IN_PADS[5]       ---       ---         ---       ---      PATH
IN_PADS[6]       ---       ---         ---       ---      PATH
IN_PADS[7]       ---       ---         ---       ---      PATH
IN_PADS[8]       ---       ---         ---       ---      PATH
IN_PADS[9]       ---       ---         ---       ---      PATH
LOAD             ---       ---         ---       ---      PATH
M1               ---       ---         ---       ---      PATH
M2               ---       ---         ---       ---      PATH
MSKCON           ---       ---         ---       ---      PATH
OUTCON           ---       ---         ---       ---      PATH
REFCON           ---       ---         ---       ---      PATH
```

```
SERIAL_IN         ---       ---            ---       ---       PATH
SPCON             ---       ---            ---       ---       PATH
TESTIN            ---       5.0            ---       0.1       PATH


**************************************************************
       Genesil Version v7.1 -- Thu Jun  7 17:32:26 1990
Chip: ~genpooler/pooler/DFT_CHIP
   Timing Analyzer
**************************************************************
VIOLATION MODE
--------------------------------------------------------------
Fabline: VTI_CN20A                    Corner: GUARANTEED
  Junction Temperature:75 deg C       Voltage:5.00v
  External Clock: CLOCK2
 Included setup files:
 #0  CLOCK2_defaults      (CLOCK2 5V 75 degree Cent results)
--------------------------------------------------------------
                     NO VIOLATIONS
  Hold time check margin: 4.0ns
--------------------------------------------------------------
```

APPENDIX D. TEST VECTOR CONVERSION PROGRAM

This Appendix provides the C source code for the conversion program used to translate Genesil MASM formatted test vector files to the ".das" and ".sim" file formats used by the DVS50 software. The conversion program also interactively produces a ".src" file for use with the DVS50 software if the conversion program user indicates that this needs to be done.

```
/*****************************************************************
convert.c -   conversion program to take the Genesil ATG or
              SIMULATOR produced MASM test vector results from
              .SMO file format and translate it to the .DAS
              and .SIM formats used by the DVS50 software
              during testing on the DAS 9100 tester.
              Additionally, a .SRC file for use by the DVS50
              software is produced interactively if desired
              by the conversion program user.
*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
#define MAXLEN 50      /* max length of signal names or test
                          vectors allowed */
#define MALLOC(x) ((x *) malloc(sizeof(x)))
#define ISWHITE(x) (x==' ') || (x=='\n') || (x=='\t') ||
                   (x=='\r')

FILE *fp1,*fp2,*fp3;   /* file pointers */
char buffer[MAXLEN+1];     /* spot to store word/testvestors
                              strings */
char spaces[MAXLEN+1];     /* holds spaces */
int count;                 /* loop counter */

main(argc,argv)
int argc;
char **argv;
```

```c
{
char temp[MAXLEN + 1];    /* used during string manipulations
                             where more than one copy of the
                             buffer information is needed */
char src_filename[MAXLEN + 1];      /* holds prefix for
                                       filename of .src file */
int undet_count = 0;  /* used to keep track of number of
                         testvectors which have undetermined
                         output values during the initialization
                         process for the chip */
int undet_line = 0;   /* line number of last line with
                         undetermined outputs*/
int vec_count = 0;    /* total number of testvector lines  */
int pin_count_das = 0;    /* counts number of input pins on
                             chip for das file */
int pin_count_sim = 0;    /* counts number of total pins on
                             chip for sim file */
long int das_count,sim_count; /* pointers to location where
                                 pin counts are going to be
                                 placed in the das and sim
                                 files*/
int loops    /* keeps track of number of multiple signals
                written during  calls to the print_to_file
                function */
if (argc == 1)    /* case of no arguements given for
                     execution*/
    {
    print_instructions();    /* print instructions and then
                                exit program */
    exit(0);
    }
else if (argc == 2)   /*  only a testvector file given as an
                          arguement */
    {
    if ((fp1=fopen(argv[1],"r")) == NULL)
        printf("Testvector file not found!");
    strcat((strncpy(buffer,argv[1],
            strcspn(argv[1],"."))),'\0');
        /* obtain the prefix charcters for argv[1] and put
        in buffer */
    strcpy(src_filename,buffer);  /* copy filename prefix*/
    fp2=fopen(strcat(strcpy(temp,buffer),".das"),"w");
        /* open a file whose name is (argv[1] prefix).das */
    fp3=fopen(strcat(strcpy(temp,buffer),".sim"),"w+");
        /* open a file whose name is (argv[1] prefix).sim */
    fprintf(fp2,"%s\n1\n1\n",buffer);  /* print das file
        prefix and two 1's on seperate lines in newly opened
        file */
    fprintf(fp3,"%s\n1\n1\n",buffer);  /* print sim file
        prefix and two 1's on seperate lines in newly opened
        file */
    }
```

```
else if (argc == 3 ) /* both testvector file and output
                        filename given as arguements for program
                        execution */
    {
    if ((fp1=fopen(argv[1],"r")) == NULL)
        printf("Testvector file not found!");
    strcpy(buffer,argv[2]);  /*set buffer=filename prefix
                                for .DAS output file*/
    strcpy(src_filename,buffer);  /* copy filename prefix*/
    fp2=fopen(strcat(buffer,".das"),"w");   /* open a file
        whose name is argv[2].das */
    strcpy(buffer,argv[2]);  /*set buffer=filename prefix
                                for .SIM output file*/
    fp3=fopen(strcat(buffer,".sim"),"w+");   /* open a file
         whose name is argv[2].sim */
    fprintf(fp2,"%s\n1\n1\n",argv[2]); /* print das file
        prefix and two 1's on seperate lines in newly opened
        file */
    fprintf(fp3,"%s\n1\n1\n",argv[2]); /* print sim file
        prefix and two 1's on seperate lines in newly opened
        file */
    }
for (count=0; count < MAXLEN; count++)
    strcat(spaces," "); /* fill spaces string with all
                            spaces */
das_count = ftell(fp2);  /* get pointer to location of das
                            file pin count */
fprintf(fp2,"000\n");    /* temp value for number of input
                            pins for das file */
sim_count = ftell(fp3);  /* get pointer to location of sim
                            file pin count */
fprintf(fp3,"000\n");    /* temp velue for number of total
                            pins for sim file */
while (strcmp(buffer,"INPUTS") != 0)     /* loop until after
        the keyword INPUTS is encountered in the testvector
        file as you move through the test vector file header
        information */
    {
    getwords(fp1);
    }
getwords(fp1);   /* get first input signal name */
while (strcmp(buffer,"OUTPUTS") != 0)
    {
    loops = 0;  /* initialize loop counter for this word */
    pin_count_das++;    /* increment counter for pins in das
                            file */
    if (strstr(buffer,";") == NULL)     /* case of all input
        signal names except the last one */
        {
        loops = print_to_file(fp2,1,loops); /*print signal
                                                name to file */
```

152

```c
        pin_count_das += loops;    /* correct input pin
                                      count as needed */
        print_to_file(fp3,1);      /* print signal name to
                                      file */
        }
    else    /* case where last character in buffer string =
            ";" which indicates the end of the inputs in the
            test vector file */
        {
        buffer[strcspn(buffer,';')-1] = '\0';    /* shorten
            word in buffer so as not to include the ";"  */
        loops = print_to_file(fp2,1);
        pin_count_das += loops;    /* correct input pin
                                      count as needed */
        print_to_file(fp3,1);
        }
    getwords(fp1);
    }
getwords(fp1);    /* get the first output signal name */
pin_count_sim = pin_count_das;      /* sim file has same
        number of input signals as the das file does */
while (strcmp(buffer,"CODING") != 0)
    {
    loops = 0;
    pin_count_sim++;      /* increment total pin count for sim
                        file */
    if (strstr(buffer,";") == NULL)      /* case of all input
            signal names except the last one */
        {
        loops = print_to_file(fp3,1);   /* print signal name
                                        to file */
        pin_count_sim += loops;
        }
    else    /* case where last character in buffer string =
            ";" which indicates the end of the outputs in
            the test vector file */
        {
        buffer[strcspn(buffer,';')-1] = '\0';    /* shorten
            word in buffer so as not to include the ";"  */
        loops = print_to_file(fp3,1);
        pin_count_sim += loops;
        }
    getwords(fp1);
    }
getwords(fp1);
while (buffer[0] != '\0')
    {
    if (strstr(buffer,"<") != NULL)    /* beginning of
                                        input test vectors*/
        {
        vec_count++;      /* increment count of number of test
                        vectors */
```

153

```c
        strcpy(buffer,&buffer[1]);       /* remove leading
                                            "<"  */
        print_to_file(fp2,1);       /* write values to file*/
        print_to_file(fp3,2);       /* write values to file*/
        }
    if (strstr(buffer,">") != NULL)      /* beginning of
                                            output testvectors*/

        {
        strcpy(buffer,&buffer[1]);       /* remove leading
                                            ">"  */
        buffer[strcspn(buffer,';')-1] = '\0';      /* shorten
            word in so as not to include the ";"  */
        print_to_file(fp3,1);       /* write values to file*/
        if (strstr(buffer,".") != NULL)
            {
            undet_count++;   /* increment count for
                undetermined output test vector cases */
            undet_line=vec_count; /* keep track of last
                test vector line that has undetermined
                outputs in it */
            }
        }
    getwords(fp1);
    }
fseek(fp2,das_count,0);  /* go to input pin count location of
                            das file */
fprintf(fp2,"%-3d",pin_count_das); /* print input pin count
                                to das file */
fseek(fp3,sim_count,0);  /* go to total pin count location of
                            sim file */
fprintf(fp3,"%-3d",pin_count_sim); /*print total pin count
                                to sim file*/
printf("\nThere are %d vectors which have undefined outputs
                    ",undet_count);
printf("caused by the\ninitialization process.
                            The last testvector with an ");
printf("undefined output\npresent was number %d.
                            The total ", undet_line);
printf("number of testvectors converted is %d.\n", vec_count);
create_src(fp3,src_filename,pin_count_sim);  /* create .src
        file if desired*/
}


/***********************************************************
create_src() -   function to create an .src file upon a
                 positive response to a query as to the
                 desire to do so.
***********************************************************/

create_src(fpointer1,filename,pins)
```

154

```
FILE *fpointer1; /* pointer to the file being read from */
int pins;        /* number of pins present on the chip */
char filename[MAXLEN + 1];    /* filename prefix for .src
                              file */
{
FILE *fpointer2; /* pointer to file to write to */
int pin_num; /* number of specific pin being referred to */
char att;  /* character representing attributes of the pin */
int name_len;    /* length of the pin name */
int pwrsp_num;   /* reference number for the power supply */
int pwrsp_count; /* counter for the number of different power
                 supplies used */
int pwrsp_voltage;   /* voltage of a particular power supply
                     in mV */
int pwrsp_current;   /* current sourced by a particular power
                     supply in mA */
int clock_rate;  /* clock rate for the application of test
                 signals */

printf("\n\nDo you want to create an .src file (Y/N)? ");
att=getche();
if (att=='N' || att=='n')      /* case where .src file is not
                               desired */
    {
    printf("\n\nProgram Execution Completed:
                    .das and .sim files created.\n");
    exit(0);
    }

/*if .src file is desired then get the needed information
about each pin*/
fpointer2=fopen(strcat(filename,".src"),"w+");     /* open a
    file whose name is (argv[1] prefix).src */
printf("\n\nFor each signal input the pin number and the
                                letter P,A, or B");
printf("\nindicating PAT, ACQ or BOTH for the pin
                                attribute.");
flushall();
rewind(fpointer1);   /* reset file to its beginning so it can
                     be read from */
getwords(fpointer1); /* get name of chip from file */
fprintf(fpointer2,"PROGRAM %s;\nPINDEF;\n",buffer);
    /* write initial info */
for (count=1; count<4; count++)    /* skip unneeded info in
                                   read file */
    getwords(fpointer1);
for (count=1; count<pins+1; count++)      /* for each pin get
        info and write it to the new file*/
    {
    getwords(fpointer1);
    printf("\n\nInput the pin number and attribute for the");
    printf(" %s signal: ",buffer);
```

```c
        scanf("%d %c",&pin_num,&att);
        name_len=strlen(buffer);
        strncat(buffer,spaces,(15-name_len));
        if (att=='P')
            fprintf(fpointer2,"%s : %2d,
                                PAT;\n",buffer,pin_num);
        else if (att=='A')
            fprintf(fpointer2,"%s : %2d,
                                ACQ;\n",buffer,pin_num);
        else if (att=='B')
            fprintf(fpointer2,"%s : %2d,
                            PAT,  ACQ;\n",buffer,pin_num);
        }

printf("\n\nFor each power supply pin enter the pin name,
                                    pin number and the");
printf("\npower supply number the pin should be connected to.
                                    When there");
printf("\nare no more power supply pins to enter information
                                    for type Z");
printf("\nfollowed by an enter.");
pwrsp_count=0;
while (1)     /* get info about power supply pins */
        {
        printf("\n\nEnter the pin name, pin number and power
                                    supply number:\n");
        scanf("%s",&buffer);
        if (strcmp(&buffer[0],"z")==0 ||
                            strcmp(&buffer[0],"Z")==0)
            break;
        scanf("%d %d",&pin_num,&pwrsp_num);
        if (pwrsp_count < pwrsp_num)
            pwrsp_count=pwrsp_num;
        name_len=strlen(buffer);
        strncat(buffer,spaces,(15-name_len));
        fprintf(fpointer2,"%s : %2d,
                        PS %d;\n",buffer,pin_num,pwrsp_num);
        }
fprintf(fpointer2,"END;\nTIMEDEF;\n");

/* get clock rate info to be used in testing */
printf("\n\nEnter the Pattern Generator clock rate in ns
                                (from 40 to 5000): ");
scanf("%d",&clock_rate);
fprintf(fpointer2,"PAT   : ns
                    %d;\nEND;\nTHRESHOLD;\n",clock_rate);

/* get threshold level for chip pins */
printf("\n\nEnter the acquisition threshold level,
                                    TTL or ECL: ");
scanf("%s",&buffer);
fprintf(fpointer2,"ACQ   : %s;\nEND;\nPSDEF;\n",buffer);
```

156

```c
/* input power supply characteristics */
printf("\n\nFor each power supply enter the desired voltage in
                                        mV and if the");
printf("\nvoltage is not 0 (ground) enter the amount of
                                        current which needs");
printf("\nto be sourced from the power supply in mA
                                        (3000 max).");
for (count=1; count<pwrsp_count+1; count++)
     {
     printf("\n\nFor power supply %d enter the needed voltage
                                        (mV): ",count);
     scanf("%d",&pwrsp_voltage);
     if (pwrsp_voltage != 0)
          {
          printf("\nEnter the max current to be sourced
                                        (mA): ");
          scanf("%d",&pwrsp_current);
          fprintf(fpointer2,"%d   : mV %d, mA
               %d;\n",count,pwrsp_voltage,pwrsp_current);
          }
     else
          fprintf(fpointer2,"%d   : mV 0;\n",count);
     }
fprintf(fpointer2,"END;\nBEGIN;\nEND$\n");

printf("\n\nProgram Execution Completed:
                         .das, .sim, and .src ");
printf("files created.\n");
}




/*************************************************************
print_to_file() -     function to print to the designated file
                      in one of two modes. Mode 1 prints the
                      contents of the buffer followed by a \n
                      (LF/CR). Mode 2 prints only the contents
                      of the buffer with no \n included.
*************************************************************/

print_to_file(filepointer,mode)

FILE *filepointer;
int mode;     /* indicates mode for output to file (1 = with
              \n, 2 = no \n) */


{
int i;
char temp[MAXLEN + 1];
int loop_count;  /* keeps track of number of iterations
                 through loop */
```

```c
if (mode == 1)    /* print string followed by \n mode */
    {
    loop_count = 0;
    if (strstr(buffer,"[") == NULL)    /*input or output
        signal names which do not have multiple signals */
        {
        if (strcmp(buffer,"CLOCK")==0)
            strcat(buffer,"_SYS");        /* identify system
                                        clock more clearly*/
        fprintf(filepointer,"%s\n",buffer); /* print string
                                        to file */
        }
    else    /*case where input/output signal names do have
            multiple signals */
        {
        loop_count--;    /* correct loops counter to start at
                        -1 */
        strcpy(temp,buffer); /* make copy of signal name */
        temp[strcspn(temp,"[")] = '\0';        /* remove
                                signal name numbers*/
        for (i=atoi(&buffer[strcspn(buffer,"[")+1]);
          i > (atoi(&buffer[strcspn(buffer,":")+1])-1); i--)
            {
            loop_count++;        /* keep track of total signals
                                printed via counting number of
                                iterations of for loop */
            fprintf(filepointer,"%s%d\n",temp,i);
                /* for each number for a given signal name
                print to the file in the format like
                SIGNALNAME_NUMBER */
            }
        }
    return(loop_count);
    }
if (mode == 2)
    fprintf(filepointer,"%s",buffer);
}


/******************************************************************
getwords() - function for getting words from the input file
            to store as appropriate in the output files.
            Note: words can consist of any printable
            character except a comma (since commas are used
            without spaces in the testvector files produced
            by ATG to seperate the input/output names).
******************************************************************/

getwords(filepointer)
FILE *filepointer;
{
int buflength,c;
```

```c
buflength=0;
strcpy(buffer,spaces);    /* fill buffer with spaces */
while (1)    /* loop until break occurs */
     {
     c=getc(filepointer);      /* get next character in file
                                 as integer value */
     if (c==EOF) /* reached end of file case */
          {
          buffer[0]='\0'; /*put null terminator in first spot
                          of buffer to serve as a flag that
                          EOF encountered */
          break;
          }
     if ((' '<c) && (c<0175) && (c != ','))
          {
          buffer[buflength++] = c;   /* if printable character
                                    except a comma place it
                                    in the buffer */
          }
     if  ((buffer[0] != ' ') && ((ISWHITE(c)) || (c==',')))
          {
          buffer[buflength] = '\0';  /* if whitespace
               character reached place null terminator at
               end of buffer string */
          if (strstr(buffer,"(") != NULL)     /* return only
               that portion of the word that does not
               contain any parenthesis */
          buffer[strcspn(buffer,"(")] = '\0';
          break;
          }
     }
}


/*****************************************************************
print_instructions() -    function to print usage instructions
                          if the program is executed without
                          any arguements
*****************************************************************/

print_instructions()
{
printf("\nCONVERSION PROGRAM TO GO FROM GENESIL .SMO TO TESTER
                                   .DAS AND .SIM");
printf("\n       FORMATS AND TO PRODUCE A .SRC FILE IF IT IS
                                   DESIRED");
printf("\n\nTo use this program call it with the name of the
                                   Genesil test");
printf("\nvector SMO file as a first arguement.  If a second
                                   arguement is");
printf("\nincluded during the call to the program then the DAS
                                   and SIM file");
```

```
    printf("\nresults will be stored under the name
                            <second arguement>.DAS and");
    printf("\n<second arguement>.SIM.  If no second arguement is
                            included then");
    printf("\nthe results will be stored under the same prefix
                            name as the SMO");
    printf("\ntest vector file.  The .SRC file is produced only if
                            desired.\n\n");
}
```

LIST OF REFERENCES

1. Frank F. Tsui, LSI/VLSI Testability Design, McGraw-Hill Book Company, 1987.

2. Thomas W. Williams and Kenneth P. Parker, "Design for testability - a survey," Proc. IEEE, vol. 71, pp. 98-112, January 1983.

3. John Stressing, "Fault simulation and test gereration - an overview," Computer-Aided Engineering Journal, vol. 6, no. 3, pp. 92-98, June 1989.

4. Jacob A. Williams, "Fault modeling in VLSI," in VLSI Testing, T. W. Williams ed., pp. 1-27, Elsevier Science Publishers B. V., 1986.

5. Tulin Erdim Mangir, "Sources of failures and yield improvement for VLSI and restructable interconnects for RVLSI and WSI: Part I - Sources of failures and yield improvements for VLSI," Proc. IEEE, vol. 72, pp. 690-708, June 1984.

6. Richard Goering, "Fault simulation strives for designer acceptance," Computer Design, vol. 26, no. 1, pp. 37-44, 1 January 1987.

7. John Carl Davidson, "Implementation of a design for testability strategy using the Genesil Silicon Compiler," Master's Thesis, Naval Postgraduate School, Monterey, California, 1989.

8. T. W. Williams, "Design for testability," in VLSI Testing, T. W. Williams ed., pp. 95-160, Elsevier Science Publishers B. V., 1986.

9. Richard Goering, "CAE and ATE vendors tighten link between design and test," Computer Design, vol. 24, no. 6, pp. 54-65, 1 October 1985.

10. Genesil System, Volume II, Parallel Data Module, Silicon Compiler Systems Corporation, San Jose, California, September 1988.

11. Dave Johannsen and Dennis G. Sabo, "Genesil silicon compilation and design for testability," 3rd International IEEE VLSI Multilevel Interconnection Conference, pp. 372-380, 1986.

161

12. _Genesil System, Volume IlI, Parallel Data Module_, Silicon Compiler Systems Corporation, San Jose, California, September 1988.

13. Robert Howard Settle, "Design methodology using the Genesil Silicon compiler," Master's Thesis, Naval Postgraduate School, Monterey, California, 1988.

14. _Genesil System Release Notes_, Silicon Compiler Systems Corporation, San Jose, California, February 1988.

15. _Genesil System Description Users Manual_, Silicon Compiler Systems Corporation, San Jose, California, September 1987.

16. _Genesil System, Compiler Library, Volume I, Blocks_, Silicon Compiler Systems Corporation, San Jose, California, February 1988.

17. _Genesil System, Simulation Users Guide_, Silicon Compiler Systems Corporation, San Jose, California, September 1987.

18. _Genesil System, Timing Analysis Users Guide_, Silicon Compiler Systems Corporation, San Jose, California, February 1988.

19. _Genesil System, Automatic Test Generation Users Guide_, Silicon Compiler Systems Corporation, San Jose, California, April 1989.

20. _MOSIS Users Manual, Release 3.0_, Information Science Institute, University of Southern California, Marina del Rey, California, 1988.

21. _DAS 9100 Series Operator's Manual with Options, Volume I_, Manual #070-3624-01, Tektronix, Inc., Beaverton, Oregon, August 1986.

22. _DAS 9100 Series Operator's Manual with Options, Volume II_, Manual #070-5396-00, Tektronix, Inc., Beaverton, Oregon, October 1986.

23. Walter F. Corliss II, "An engineering methodology for implementing and testing VLSI circuits," Master's Thesis, Naval Postgraduate School, Monterey, California, 1989.

24. _91DVS Device Verification Software User's Manual_, Manual #070-6072-00, Tektronix, Inc., Beaverton, Oregon, September 1986.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Commandant of the Marine Corps 1
   Code TE 06
   Headquarters, U. S. Marine Corps
   Washington, D.C. 20380-0001

4. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Dr. Herschel H. Loomis, Code EC/LM 5
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Dr. Chyan Yang, Code EC/YA 3
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

7. Captain Brian L. Pooler 1
   Electrical Engineering Department
   U. S. Naval Academy
   Annapolis, Maryland 21402